Are Object Graphs Extracted Using Abstract Interpretation Significantly Different from the Code?

Marwan Abi-Antoun
Sumukhi Chandrashekar
Radu Vanciu
Andrew Giang

Wayne State University
Department of Computer Science
Detroit, Michigan, USA
Motivation: Runtime structure

- Representing a system
  - Static/Code structure (class diagram)
  - Dynamic/Runtime structure

Class diagram do not answer
- What kind of architecture does system follow?
- Do PieChart, BarChart, Model share one Listener?
- Are different ArrayList instances conceptually different?
Motivation: Runtime structure

- Representing a system
  - Static/Code structure
  - Dynamic/Runtime structure
    - Eclipse debugger
      - Wade through many instances
      - Specific instances may not matter for some tasks
    - Object graphs
      - Too large (without abstraction)
      - May not convey design intent
      - Need to apply abstraction
How to apply abstraction?

- Merge related objects
- Collapse objects underneath other objects
Abstracted heaps or object graphs

• Dynamic analysis
  • Abstractions of dynamically extracted heaps
    [Marron et al., TSE, 2013] [Barr et al., ISSTA, 2013]

• Static analysis
  • Abstract interpretation to approximate an abstract runtime structure (OGraph)
    [Abi-Antoun and Aldrich, OOPSLA, 2009]
Abstract Runtime Structure (OGraph)

OGraph expresses:
- abstract objects (can have multiple of the same type)
- abstract edges for runtime relations between abstract objects
- Hierarchy: object ->* domain -> * object

OGraph = global, hierarchical points-to graph extracted using whole program analysis
Merge objects that play the same role

- Role of an object described by:
  - Type of object
    - Type A
  - In domain
    - Domain D
  - And position in object hierarchy
    - Parent object of Type B
  - \(<A, D, B>\) triplet

Hierarchy of objects:
- sys: Main
  - DOCUMENT
    - m: Model
      - owned
        - I: ArrayList<Listener>

In parent object of Type B
Inside Domain D
Object of Type A
Why corpus analysis?

• Previous evaluations of runtime structure:
  • Case studies, field study
  • Controlled experiment

• Runtime structure seems to help:
  [Quante, ICPC, 2008] [Ammar and Abi-Antoun, WCRE, 2012]
  • For some systems, but not others
  • For some tasks, but not others

• Need to better understand:
  • Differences between code and abstract runtime structure
  • Code patterns that lead to greater differences
  • Comprehension difficulties addressed by runtime structure
Outline

• Compare runtime with code structure
• Research questions
  • Metrics
  • Examples of code identified by metrics
• Outliers & analysis of outliers
  • Transcript analysis
  • Classify outliers
Contributions

• *Define metrics* measuring differences between code and abstract runtime structure

• *Quantitative statistical analysis* of metrics across 8 subject systems (totaling 100 KLOC)
  - *p-value* based on one-sample Wilcoxon non-parametric test
  - magnitude of differences using *Cliff’s Delta*

• Qualitative analysis of outliers to identify code patterns that contribute to greater differences

• Transcript analysis to relate outliers to program comprehension difficulties
Metrics

• Relate *one* code element to *many* abstract runtime elements
• Relate *many* code elements to *one* abstract runtime element
• Relate *syntactic* with *semantic* location of elements
• Compare *precision* of information from runtime structure to information from code structure
RQ1: one-to-many

RQ1: How many abstract runtime elements correspond to one code element?

Metric: One Field Declaration Many Edges (1FnE)

Definition: Measures how many edges in the OGraph are due to the same field declaration in the code
RQ2: many-to-one

RQ2: How many code elements correspond to one abstract runtime element?

**Metric:** Different New Expressions Same Object (HMN)

**Definition:** Measures how many distinct object creation expressions new A() are abstracted by the same OObject OA

Code

```java
@Domains("DOCUMENT","VIEW")
class Main {
    @Domain("VIEW<..>") BarChart b = new BarChart();
}

@Domains("owned")
@DomainParams("M","V")
@DomainInherits("BaseChart<M,V>")
class BarChart extends BaseChart {
    @Domain("V<M,V>") BarChart b1 = new BarChart();
}
```

Hierarchy of objects

```
| • sys: Main
|   • VIEW
|      • b: BarChart
|      • p: PieChart
```
RQ3: mismatch

**RQ3:** How often does the location of an abstract runtime element mismatch the location of its corresponding code element?

**Metric:** Pulled Objects (PO)

**Definition:** An object of type A may be pulled into a domain D that is inside some parent object of type B. Measures the percentage of pulled objects compared to all objects in the OGraph.

```java
@Domains({"DOCUMENT","VIEW"})
class Main {
}
@Domains({"owned"})
@DomainParams({"M","V"})
@DomainInherits({"BaseChart<M,V>"})
class BarChart extends BaseChart{
    @Domain("V<M,V>") BarChart b1 = new BarChart();
}
```

Hierarchy of objects

```
|- sys: Main
  |+ DOCUMENT
  |- VIEW
    |+ b1: BarChart
    |+ p: PieChart
```
RQ4: precision

RQ4: How often does the abstract runtime structure have more precision than the code structure?

Metric: Points-to Edge Precision (PTEP)

Definition: Measures how precisely the OGraph reveals concrete types of the abstract objects hiding behind a field of an abstract type

Code

```java
@DomainParams(ELTS)
class ArrayList<E>
{
    @Domain("ELTS") E value;
}

@Domains("owned") @DomainParams(M,V)
@DomainInherits(BaseChart<M,V>)
class BaseChart extends Listener
{
    @Domain("owned<M>") List<Listener> l = new ArrayList<Listener>;
}

@Domains(DOCUMENT,VIEW)
class Main
{
    @Domain("DOCUMENT<DOCUMENT,VIEW>")
    Model m = new Model();
}
```

OGraph

- m: Model
- b: BarChart
- p: PieChart
- l: ArrayList<Listener>
- VIEW

Subtypes shown by OGraph

1. simple.Model
Outliers

• Computed metrics on 8 subject systems
• Examples here taken from MiniDraw (MD)
• Identified outliers
• Analyzed outliers
  • Relate to program comprehension difficulties (*Transcript analysis*)
  • Identify code patterns (*Classify outliers*)
Transcript Analysis

- **Previous experiment** [Ammar and Abi-Antoun, WCRE, 2012]
  - 10 participants, split into 2 groups
    - Control group (C): used only class diagrams
    - Experimental group (E): used OGraphs
  - 3 tasks on MiniDraw (MD)
  - Captured logs (Transcripts)
- Re-analyzed transcripts
Transcript Analysis: 1FnE

- Task: Restrict the piece movements
- Struggle with: Code that implements methods to move pieces

**Code**

```java
class BoardActionTool extends AbstractTool {
    // Moves pieces when mouse is clicked on the square below current
    void mouseDown() {
    }
    // Moves pieces when mouse is dragged
    void mouseDrag() {
    }
    // Moves pieces when mouse is clicked on the square above current
    void mouseUp() {
    }
}
```

**OGraph**

```
bat: BoardActionTool
   ——
   \——
    ——
      bf: BoardFigure
      bd: BoardDrawing
```

**Logs from the participants:**
C1: "And where do they actually perform the movement of pieces?"
C3: "So we need to find out where the pieces move and who moves it, but where!?"
Transcript Analysis

• Mining transcripts indicate
  • Code associated with outliers is often explored
  • Outliers point to program comprehension difficulties
Analyzed Outliers

- Analyzed outliers
  - Relate to program comprehension difficulties (*Transcript analysis*)
  - Identify code patterns (*Classify outliers*)
Code Patterns

• Code traced from outliers follows patterns:
  • Container of general type (predefined list of containers)
  • Field of general type (interface or abstract class)
  • Inheritance
  • Composition
Code Pattern: Inheritance

Result: many edges from object of enclosing type of field to objects of subtypes of field type

code:

```java
class B {
    A a = new C(); // Field of general type
}

abstract class A {
}
class C extends A {
}
class D extends A {
}
```

OGraph:

```
c: C

b: B

a

---
d: D
```
Classify Outliers: 1FnE

Study system: MD
Classification bucket: Inheritance

Type hierarchy of BoardDrawing
BoardDrawing <: StdDrawing <: CompositionFig <: AbstractFigure <: Figure

Type hierarchy of BoardFigure
BoardFigure <: ImageFigure <: AbstractFigure <: Figure

Code

```java
@Domains({"owned"})
@DomainParams({"U","L","D"})
class SelectionTool {
    @Domain("D<U,L,D>") Figure draggedFig = null;
}
class BoardFigure extends ImageFigure {
}
class BoardDrawing extends StdDrawing {
}
```

OGraph

```
selt: SelectionTool
    draggedFig
        bf: BoardFigure
draggedFig
    bd: BoardDrawing
```
Related Work

• Statically abstract object graphs extracted using annotations [Lam and Rinard, ECOOP, 2003]

• Metrics on dynamically extracted abstract object graphs [Barr et al., ISSTA, 2013]

• Metrics of the code structure or of the runtime structure
  [Arisholm et al., TSE, 2004], [Bavota et al., ICSE, 2013]

• Metrics to evaluate different context-sensitivity
  [Kastrinis and Smaragdakis, PLDI, 2013]
Conclusions

• Differences between code and runtime structure of systems are small but statistically significant
• Better understanding of systems for which extracting runtime structure worthwhile
• Better understanding of code patterns that contribute to larger differences
• Better understanding of program comprehension difficulties in object-oriented code
For more information

• See the paper
• Online appendix with data
  http://www.cs.wayne.edu/~mabianto/arch_metrics/
• Technical report with formalization of metrics
  • how we handle inheritance, aliasing, etc.
• Details of our implementation
  • Plots generated from R