The Eclipse Runtime Perspective for Object-Oriented Code Exploration and Program Comprehension

Marwan Abi-Antoun
Andrew Giang
Sumukhi Chandrashekar
Ebrahim Khalaj

Wayne State University
Department of Computer Science
Detroit, Michigan, USA
Motivation: A new Perspective

• IDEs emphasize design-time perspective based on code structure
  • Class-oriented view
  • Hierarchy of classes

Such views do not answer
• What are the architectural tiers of the application?
• Where is an instance of type A created?
• Can one access an instance of type A from an instance of type B?
Motivation: A new Perspective

- Another perspective: the runtime perspective
  - Eclipse debugger
    - Specific instances of type A may not matter
  - Object graphs:
    - Too large (without abstraction)
    - May not convey design intent
    - Need to apply abstraction

Flat object graph: Womble [Jackson and Waingold '01]
Motivation: A new Perspective

• Emphasize design-time perspective based on **abstract** runtime structure
  • Extracted using abstract interpretation
  • Hierarchy of abstract objects
    [Abi-Antoun and Aldrich, OOPSLA, 2009]
  • Summarization of runtime objects
  • Abstraction keeps graph manageable
• Use static analysis so tool works at design time
Motivation: A new Perspective

- New perspective integrated with Eclipse IDE
Contributions

• Novel Eclipse development-time perspective
  • focuses on abstract runtime structure
  • complements existing perspectives
  • displays information to developers using diagrammatic and non-diagrammatic views
Outline

• Extract abstract runtime structure
• Task centric demonstration of ArchDoc
• Contrast Java perspective with ArchDoc
• Potential applications of ArchDoc
Extracting abstract runtime structure

- Add annotations and type check them
  - Express architectural hierarchy
  - Annotations must match the code
- Extract hierarchical abstract object graph using static analysis
  - Architecturally relevant objects at the top level
  - Abstract edges connect abstract objects
- Save abstract graph to external file
- Switch to Runtime Perspective (ArchDoc)
  - Mines the hierarchical abstract graph
  - Displays information in various view
Java vs. ArchDoc

- **Java Perspective**
  - Package Explorer
  - File/Java Search
  - Type Hierarchy
  - Call Hierarchy
  - Class Diagrams

- **ArchDoc**
  - Abstract Object Tree
  - Abstract Object Search
  - Related Objects and Edges
  - Abstract Stack
  - Partial Graph View
Task on MiniDraw (MD)

- Validate piece movements on board game

- Pieces move to non-empty squares only
- Direction of movement is straight or diagonal
- Direction of movement has to be diagonal if capture of opponent pieces
Developer Questions

• Q1: What are the architectural tiers?
• Q2: Where is canonical object of type A created?
• Q3: Can one retrieve related types/objects of type A?
• Q4: Can one access a canonical object of type A from type B?
• Q5: What are the concrete types of a canonical object of type A at runtime?
Q1: What are the architectural tiers?

- Hierarchy of classes organized into packages
- Packages, classes or interfaces sorted alphabetically
- Tiers not visible in Java code
Q1: What are the architectural tiers?

- Hierarchy of abstract objects and domains
- Top-level domains are the architectural tiers
Q1: What are the architectural tiers?

- Shows that MD follows 3-tiered style
- Developers can focus on objects in CONTROLLER or MODEL for this task
Q2: Where is canonical object of type A created?

Eclipse Java Search

Shows results from:
- Comments
- Declaration points
- Usage points
- Java libraries
Q2: Where is canonical object of type A created?

- Search for abstract objects by type, name
- Trace back to object creation expressions
Q2: Where is canonical object of type A created?

- Instances of type BoardFigure is created in BoardDrawing
- Board pieces are ArrayList<BoardFigure>
Does abstract runtime structure differ from the code structure?

E.g., do abstract objects of the same type appear in different parts of the object tree?

[Abi-Antoun et al., SCAM, 2014]

Trace to code

```java
@Domains({"owned"})
@DomainParams({"U","L","D"})
@DomainInherits({"Figure<U,L,D>"})
abstract class AbstractFigure implements Figure {
    @Domain("owned<D<U,L,D>>") List<FigureChangeListener> listenerList;
    public AbstractFigure() {
        listenerList = new ArrayList<FigureChangeListener>();
    }
}
```
Q3: Can one retrieve related types/objects of type A?

Code Fragment

@Domains("owned")
@DomainParams({"U","L","D"})
@DomainInherits{"Command<U,L,D>"}
class MoveCommand implements Command {
    @Domain("D") Position VIRTUAL_from = Null;
    ...
}
Q4: Can one access a canonical object of type A from type B?

- Hierarchy in abstract object tree
- Objects representing data structures are beneath relevant objects
Q4: Can one access a canonical object of type A from type B?

ArchDoc Hierarchical Abstract Object Graph

Legend
- Domains
- Abstract objects
- Points-to
Q5: What are the concrete types of a canonical object of type A at runtime?

- Shows all possible subtypes of Figure
- Including non concrete types
Q5: What are the concrete types of a canonical object of type A at runtime?

- Interested only in concrete types of Figure declared in BoardActionTool
- Refers to 2 points-to edges from BoardActionTool
Eclipse Call Hierarchy

• Shows caller and callees transitively for a selected method
• Traces to method invocations
ArchDoc Abstract Stack

- Contrast with Call Hierarchy
- Shows object creation hierarchy
- Shows abstract interpretation contexts
- Exposes notion of "object sensitivity"
Applications

- Explaining Design Patterns
- Explaining Shallow vs. Deep Copy
Explaining Design Patterns

• Observer design pattern.

Logical Model

- Observer: BoardDrawing
- Subject: GameStub
Explaining Design Patterns

• Observer design pattern.

Code Fragment

```java
@Domains({"owned"})
@DomainParams({"U","L","D"})
@DomainInherits({"Game<U,L,D>"})
class GameStub implements Game {
    @Domain("D<U,L,D>") BoardGameObserver observer;
    void move(@Domain("D") Pos from, @Domain("D") Pos to) {
        observer.pieceMovedEvent(from, to);
    }
}

BoardDrawing <: BoardGameObserver
```

ArchDoc Hierarchical Abstract Object Graph
Related tools

• Objektgraph: flat and non abstract depictions of runtime structure
  [Buck et al., SPLASH, 2013]

• Code exploration: focus is on code structure
  [Kollman et al., WCRE, 2002]

• Call graph: focus is on visualizing call graphs
  [Bohnet and Döllner, WODA, 2006]

• Heap exploration: focus is on visualize and interactively explore snapshots of the heap
  [Kelley et al., Information Visualization, 2012]
Conclusion

- ArchDoc complements existing design-time perspective in Eclipse
- ArchDoc helps answer developer questions based on abstract runtime structure
Future work

• Evaluate the tool in user studies
  • Replicate results from previous experiment
    [Ammar and Abi-Antoun, WCRE, 2012]

• Use the tool in educational setting
  • Beginners learning design patterns, etc.
  • Use in laboratory component of course