Extraction of Ownership Object Graphs from Object-Oriented Code: an Experience Report

By: Marwan Abi-Antoun, Nariman Ammar, Zeyad Hailat

Presenter:
Radu Vanciu (advisor: Marwan Abi-Antoun)
Ph.D. Student
SoftwarE Visualization and Evolution REsearch (SEVERE)
Department of Computer Science
Wayne State University, Detroit, USA
June 2012
Extraction of Runtime Architecture is Challenging

- Runtime architecture models runtime entities (objects) and their potential interaction
  - Groups objects into tiers
  - Allows maintainers to reason about quality attributes, such as security or performance
  - Architectural diagrams often too high level, missing, or inconsistent with code
- Runtime and code architectures are equally important, complementary
Soundness of Extracted Architecture

- Allows reasoning about worst-case scenarios
- Extracted object graph approximates any Runtime Object Graph
  - Each runtime object has **exactly one** representative in extracted architecture
  - Edges correspond to all possible runtime relations between runtime objects
Use Hierarchy to Manage Complexity

Flat object graph (WOMBLE) [Jackson and Waingold '01]

Hierarchical object graph (SCHOLIA) [Abi-Antoun and Aldrich '09]

Fragment of ownership tree (SCHOLIA)
- system : System
  - LOGIC
    - f : InputFile
    - iniFile : IniFile
      - owned
        - paragraphs : Hashtable
      - PARAGS
        - para : IniParagraph

Extracted object graphs for Pathway-Express (PX) 36 KLOC
Contributions

- An experience report on:
  - Extracting OOG of medium sized object-oriented system
  - Refining OOG based on the maintainers design intent

- Others can use SCHOLIA, not only its designers

- Estimated effort: 1 hour / KLOC

- An evaluation of extracted OOG by lead maintainer (further refinements so OOG matches his mental model)

- Confirmation that refinement effort is lower than initial extraction effort
SCHOLIA for Architectural Extraction

- Supports legacy code

- Requires ownership domain annotations
  - Express architectural hierarchy
  - Domain is similar to architectural runtime tier
  - Annotations are consistent with each other and with code

- From code with annotations, static analysis extracts a global, hierarchical ownership object graph (OOG)
  - Each domain has a unique parent object
  - Each object has a unique parent domain
  - Architecturally relevant objects at the top of hierarchy
  - Implementation details in substructure of higher level objects
  - Edges between objects represent field references (points-to edges)
Annotation Examples

- Named group of objects
  @Domain: put in architectural tier

```java
class Main{
    @Domain("UI") PEInputApplet applet;
    @Domain("DATA") PathwayExpressData peData;
    @Domain("LOGIC") IniFile iniFile;
}

class IniFile{
    @Domain("owned<PARAGS>") Hashtable paragraphs;
    @Domain("PARAGS") IniParagraph para;
}

// L := LOGIC
class IniParagraph{
    @Domain("L") InputFile f;
}
```
Ownership Hierarchy

- **Strict encapsulation**
  - Avoids giving external objects access to private state of an object (avoids representation exposure)
  - `@Domain("owned")`: a public method cannot return an alias to a field in private domain

- **Logical encapsulation**
  - Allow access to all objects in public domains
  - `@Domain("PARAGS")`: access to iniFile gives access to para

- **Collapse object’s substructure**
  - Lift edges to the nearest visible ancestor
  - iniFile’s substructure is not visible
Ownership Hierarchy

- **Strict encapsulation**
  - Avoids giving external objects access to private state of an object (avoids representation exposure)
  - `@Domain("owned")`: a public method cannot return an alias to a field in private domain

- **Logical encapsulation**
  - Allow access to all objects in public domains
  - `@Domain("PARAGS")`: access to iniFile gives access to para

- **Collapse object’s substructure(+)**
  - Lift edges to the nearest visible ancestor
  - iniFile’s substructure is not visible
Architectural Extraction Process

- Code
- Annotations
- Ownership
  - Object Graph
- Type-domain map
- Analysis Settings
  - (Design Intent Types)
- Analysis Input
- Analysis Output

Extractor

Maintainer

Automatic activity

Manual activity

- trace to code
- typecheck
- refine
- extract
- merge
- generate
- inspect
- annotate
- map
- provide
- request refinements
- evaluate
Participants

- **Extractor**
  - First-year Ph.D. student (third author of the paper)
  - Received classroom training
  - Practiced on several small examples (<1.4 KLOC)

- **Maintainer**
  - Leadmaintainer of the system
  - Several years of experience with Java and Eclipse
  - Fifth year Ph.D. student
Subject System

- Pathway-Express (PX)*
  - More than 1,000 users find, build, and display graphical representations of gene interactions
  - Original developer is a former Ph.D. student (no longer involved in PX maintenance)
  - Actively maintained by graduate students (who struggle to understand the system)

- Size: 36 KLOC, excluding libraries
  - 30 packages
  - 163 classes
  - 9 interfaces

- We have access to the lead maintainer of PX

*http://vortex.cs.wayne.edu/projects.htm#Pathway-Express
Abstraction by ownership hierarchy

- Change annotations to:

**R1**: Move objects between sibling domains

**R2**: Abstract low-level objects
  - **R2.1**: private domain (owned by)
  - **R2.2**: public domain (part of)

**R3**: Move object to higher level domain
Abstraction by Types

- Change analysis settings to:

R4: Merge related instances of types that share a common super-type

- Requires a list of design intent types
- OOG merges objects in the same domain
Tool Support

- **ArchDefault**
  - Generates boilerplate annotations based on extractor specified map

- **Typechecker**
  - Ensures that annotations are consistent with each other and with code
  - Outputs warnings in Eclipse Problems Window

- **OOG extraction tool**
  - Displays OOG
  - Interactive navigation (expand / collapse objects)
  - Trace to code
Research Questions

- **RQ1**– Can extractor effectively use abstraction by ownership hierarchy and by types to extract an OOG that conveys architectural abstraction? And how much effort does it take?

- **RQ2**– Can the maintainer understand the OOG, i.e., abstraction by ownership hierarchy and by types?

- **RQ3**– Can the extractor incrementally refine the OOG to make it convey the maintainer’s design intent?
Success Criteria

- **Extractor:**
  - Reduce number of typechecker warnings
  - Minimize refactoring of code to add annotations
  - Minimize effort of adding annotations and extracting OOG

- **Maintainer:**
  - OOG that is sufficiently abstract and not too cluttered
    (rule of thumb: 5 – 7 objects per domain)
  - OOG that reflects maintainer’s design intent.
OOG Extraction Tool
Extractor – Maintainer meeting

Questions at the meeting

- **Q1**: Does object X of type T belong to tier A? And if not, to which tier does it belong?
- **Q2**: Which objects, do you think, are useful and helpful for code modifications to see at the top-level of the OOG?
- **Q3**: Are there any missing objects from the top-level of the OOG? or from the rest of the OOG?

Procedure

- Quick tutorial on how to use OOG extraction tool and visualization
- Extractor asks maintainer the above questions
- Maintainer navigates through each tier and object, one by one
- Extractor collects requested refinements
Requested Refinements

**R1**: Move object of type PETableModel from UI to DATA

**R2**: Examples of strict encapsulation and logical containment

**R2.1**: Make object of type LoginRequest owned by LoginFrame

**R2.2**: Make object of type IniParagraph part of IniFile

**R4**: Merge related instances of FunctionBar

Split PEGUIManager across two tiers
(not supported by ownership domains)

Collapse PathwayDetailsTable, InputIdGenesTable and PETableModel
(types do not share a common super-type)

<table>
<thead>
<tr>
<th>OOG Refinement</th>
<th>Requested</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 Move object between sibling domains</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>R2 Abstract low-level object</td>
<td>40</td>
<td>21</td>
</tr>
<tr>
<td>R3 Move object to higher-level domain</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R4 Collapse related instances of subtypes</td>
<td>20</td>
<td>14</td>
</tr>
</tbody>
</table>
RQ1: Can extractor effectively use OOG abstraction mechanisms?

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>objects</td>
<td>222</td>
<td>240</td>
</tr>
<tr>
<td>top-level objects</td>
<td>72 (33%)</td>
<td>59 (25%)</td>
</tr>
<tr>
<td>top-level objects after abstraction by types</td>
<td>68 (30%)</td>
<td>46 (20%)</td>
</tr>
<tr>
<td>design intent types</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
RQ1: How much effort does it take to extract an OOG?

- Effort of adding annotations and extracting initial OOG was 31 hours for 36 KLOC (1 hour / KLOC)
- Consistent with previous measurements (LBGrid: 35 hours for 30 KLOC [Abi-Antoun and Aldrich, PASTE’08])
- Meeting with maintainer was around 1 hour

<table>
<thead>
<tr>
<th>Phase</th>
<th>Effort(Hours)</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding annotations and extracting OOGs</td>
<td>31</td>
<td>69%</td>
</tr>
<tr>
<td>Building the ArchDefault map</td>
<td>5</td>
<td>11%</td>
</tr>
<tr>
<td>Refining the OOG on our own</td>
<td>5</td>
<td>11%</td>
</tr>
<tr>
<td>Meeting with maintainer</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Refining the OOG after meeting</td>
<td>3</td>
<td>7%</td>
</tr>
<tr>
<td>Total</td>
<td>45</td>
<td>100%</td>
</tr>
</tbody>
</table>
Related Question: Can we measure quality of annotations?

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Frequency</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>125</td>
<td>2.2%</td>
</tr>
<tr>
<td>L</td>
<td>75</td>
<td>1.3%</td>
</tr>
<tr>
<td>D</td>
<td>511</td>
<td>9.1%</td>
</tr>
<tr>
<td>owned</td>
<td>278</td>
<td>4.9%</td>
</tr>
<tr>
<td>shared</td>
<td>2,994</td>
<td>53.1%</td>
</tr>
<tr>
<td>unique</td>
<td>363</td>
<td>6.4%</td>
</tr>
<tr>
<td>lent</td>
<td>1,273</td>
<td>22.6%</td>
</tr>
<tr>
<td>Public domains</td>
<td>6</td>
<td>0.1%</td>
</tr>
<tr>
<td>Top-level domains</td>
<td>3</td>
<td>0.1%</td>
</tr>
<tr>
<td>Other domain parameters</td>
<td>6</td>
<td>0.1%</td>
</tr>
<tr>
<td>Total</td>
<td>5,634</td>
<td>100%</td>
</tr>
</tbody>
</table>

- High proportion of shared due to excessive use of String
RQ2: Can the maintainer understand OOG?

- Abstraction by ownership hierarchy
  - 40 requests to move an object underneath some other object
  - Overall, reduced number of top-level objects

- Abstraction by types
  - Maintainer requested to merge several objects
  - Not all the requests were supported by code. Example: classes PathwayDetailsTable, InputIdGenesTable and PETableModel do not share a common super-type

<table>
<thead>
<tr>
<th>Domain</th>
<th>At meeting</th>
<th>After meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI</td>
<td>29</td>
<td>7</td>
</tr>
<tr>
<td>LOGIC</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>DATA</td>
<td>27</td>
<td>23</td>
</tr>
<tr>
<td>Total</td>
<td>68</td>
<td>46</td>
</tr>
</tbody>
</table>
RQ3: Can extractor incrementally refine OOG?

- Extractor addressed most refinement requests without changing all annotations
- Significant less time spent in refinement vs. extraction (3 vs. 31 hours)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Effort (Hours)</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding annotations and extracting OOGs</td>
<td>31</td>
<td>69%</td>
</tr>
<tr>
<td>Building the ArchDefault map</td>
<td>5</td>
<td>11%</td>
</tr>
<tr>
<td>Refining the OOG on our own</td>
<td>5</td>
<td>11%</td>
</tr>
<tr>
<td>Meeting with maintainer</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Refining the OOG after meeting</td>
<td>3</td>
<td>7%</td>
</tr>
<tr>
<td>Total</td>
<td>45</td>
<td>100%</td>
</tr>
</tbody>
</table>
Lessons Learned

- OOG enables maintainer to identify design issues
  - Annotations revealed occurrences of representation exposure
  - Lack of rich inheritance hierarchy
  - Loosely-typed containers
  - Lack of user-defined types

- OOG can guide maintainer toward fixing design issues
  - Create common supertype for related types
    (PathwayDetailsTable, InputIdGenesTable and PETableModel)
  - Use composition instead of inheritance to place different parts of an object in different domains
    (refactor PEGUIManager into two classes)
Threats to Validity

- **Internal validity**
  - Maintainer was not familiar with all the objects in PX
  - Result is not due to the extractor expertise
    (extractor guided only by Typechecker)
  - Maintainer did not validate mapping before using ArchDefault, which may have increased the overall time

- **External validity**
  - PX not representative – written by students, not professionals
  - 500 warnings unsolved (LBGrid: 4,000 warnings)
  - Design issues in PX (e.g., representation exposure)
Limitations and Future Work

- ArchDefault is not a smart ownership inference tool
- Expressiveness challenges in ownership type system (e.g., static code)
- OOG shows only points-to edges (maintainer requested additional types of edges)
- Lack of interactive refinement of the OOG
- Future: Observe maintainers using OOG during software evolution tasks
Conclusions

- OOG extracted from code with typecheckable annotations conveys maintainer’s design intent

- Maintainer understood abstraction by ownership hierarchy and by types

- Effort required to add annotations and extract OOG is 1 hour / KLOC

- Effort required to refine OOG significantly less than effort to extract initial OOG