On Modeling Software Architecture
Recovery as Graph Matching

Kamran Sartipi

Department of Computing and Software
McMaster University
Canada
Sartipi@mcmaster.ca

IEEE International Conference on Software Maintenance (ICSM 2003)

&

Outline

- Motivation and definition for software architecture and software architecture recovery
- Issues to be addressed in a software architectural recovery environment
- Proposed approach to support reflective and pattern-based architectural recovery
- Conclusion and future research directions
Motivation for Software Architecture Recovery

- Average life-time of large systems is 10-15 years. Replacement of these systems is very expensive.
- Adopting a new technology such as: object-orientation, component-based programming, or network-centric requires changes in the design of system.
- Maintenance activities such as error-correction and feature enhancement, invalidate the design documents.
- Migrating a legacy system to a new platform such as Windows or Unix requires functional description of the system’s components.
Software Architecture

- A generally accepted definition:

  “The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time” [SEI 1994]

- However, software architecture is more than “components and connectors”, or “major elements of a system”. It is a collection of views, patterns, stakeholders, and roles [SEI].

- Therefore, Software architecture provides the necessary means to formalize and interpret the properties of a software system.
Software Architecture Recovery

Extracting high-level structural information from low-level system representation such as source-code

Major architecture recovery techniques:
- Clustering [MQ-partitioning, ACDC]
- Concept lattice analysis [Repairing, Horizontal]
- Pattern-based techniques [Dali, Recognizers]
- System visualization and analysis [Pbs, Rigi]
Issues to be addressed by an architectural recovery environment

- What view of the system to recover?
- How to represent the software system?
- How to model the high-level view of system?
- What recovery technique to use?
- How to scale the recovery process?
- How to involve the user in recovery?
- How to validate the architecture?
Graph Matching techniques

- Exact and approximate graph matching techniques:
  - Comparing primitives of prototype and input graph.
  - Decomposing the graphs into simple trees to match.
  - Generating an state space using cost of graph edit operations and search for minimum path.

- Graph in reverse engineering:
  - Adopted as standard for information exchange among tools.
  - Uniform mechanism for representing the software system and performing pattern matching process.
Environment for Pattern-based Software Architecture Recovery

Off-line: pre-process

On-line: analysis

- System analysis
- Domain & Document
- Decision making

Module-Interconnection pattern

AQL query

Query generation

Graph generation

Architecture & Evaluation

Graph matching engine (search & evaluation)

Pattern graph

Software System

C / Pascal / …

Parsing

RSF

AST

Software as graph

Data mining

Graph regions & Similarity matrix

Kamran Sartipi: ICSM 2003, Book LAMBERT Academic
Abstract domain model provides abstraction of the source-level domain model

- **Entity-types**: a subset of entity-types in source-code
- **Relation-type**: an aggregation of one or more relation-types in source-code
Domain model for software system

Entity-abs
- name: String
- file #: Integer
- line #: Integer
- implement-id: Char Integer

File-abs
- Id: 'L' Integer
- imports: set (Entity-abs)
- exports: set (Entity-abs)
- contains: set (Entity-abs)
- uses: set (Entity-abs)

Variable-abs
- Id: 'V' Integer

Type-abs
- Id: 'T' Integer

Function-abs
- Id: 'F' Integer
- useFuncs: set(Function-abs)
- useTypes: set (Type-abs)
- useVars: set (Variable-abs)

use-R
from: File-abs
to: Entity-abs

cont-R
from: File-abs
to: Entity-abs

imp-R
from: File-abs
to: Entity-abs

exp-R
from: File-abs
to: Entity-abs

use-V
from: Function-abs
to: Entity-abs

use-T
from: Function-abs
to: Entity-abs

use-F
from: Function-abs
to: Entity-abs

Relation-abs
- file #: Integer
- line #: Integer
- implement-id: Char Integer
Dividing the system graph into regions

**System representation:** the collection of source-regions

Source graph

\[ G^S = (N^S, R^S) \]

Source-region 1

Data mining

Source-region 6

Nodes = \{1, 7, 10, 2, 13, 6, 11, 16, 15\}

similarity = [4, 4, 4, 4, 3.5, 3, 3, 3]
Architecture Query Language (AQL)
Module Interconnection Pattern

MODULE: M1
MAIN-SEED: func search_class
IMPORTS:
FUNCTIONS: func ?IF, func ?F5(3..6) M2
TYPES: type ?IT, type ?T1(0..4) M3
VARIABLES: var ?IV
EXPORTS:
FUNCTIONS: func ?EF, func ?F1(2..5) M3
TYPES: type ?ET
VARIABLES: var ?EV
CONTAINS:
FUNCTIONS: func $CF(15 .. 18), func search_class (), func inherit_facts ()
TYPES: type $CT(0 .. 2)
VARIABLES: var $CV(3 .. 5)
END-ENTITY

Modeling high-level view of system

Kamran Sartipi: ICSM 2003, Book LAMBERT Academic
Domain model for AQL
(Conceptual Architecture)
Graph Matching Model of Recovery

Kamran Sartipi: ICSM 2003, Book LAMBERT Academic
Approximate graph matching

- $f: G_1 \rightarrow G_2$ maps the nodes and edges of $G_1$ onto $G_2$.
- Different forms of function $f$:
  - Homomorphism: $f$ can map two nodes of $G_1$ to one node of $G_2$.
  - Monomorphism: $f$ is one-to-one (i.e., sub-graph isomorphism).
  - Isomorphism: $f$ is one-to-one in both directions.
- Exact graph matching:
  - Identifies exact set of nodes and edges of $G_1$ that matches with $G_2$ (in most real applications is not feasible).
- Approximate graph matching:
  - An optimal sequence of graph edit operations, such as: insertion / deletion of nodes and edges of $G_1$ so that $G_1$ and $G_2$ become isomorphic.
Different types of graphs

- Source-graph: $G^s$
- Query graph: $G^q$
- Source-region: $G_{g(i)}^{sr}$
- Pattern-region: $G_{i}^{pr}$
- Input graph: $G_{i}^{l}$
- Pattern graph: $G_{i}^{p}$
- Matched graph: $G_{i}^{m}$
Modeling software architecture recovery as “graph pattern matching”

Given a query graph $G^q = (N^q, R^q)$ that is expanded to a pattern graph $G^p$,
given a system graph $G^s = (N^s, R^s)$, and
given a graph distance threshold $d_t$,
the problem is to find a sub-graph of $G^s$ i.e. $G^m$ that approximately matches with the pattern graph $G^p$, so that:

$$\text{dist}(G^p, G^m) < d_t \quad \& \quad \text{dist}(G^p, G^m)|_{\text{min}}$$
Graph algebraic model of matching process

At each phase $i$ of the matching process, $G_i$ is approximately matched against $G_i$ which results in $G_i$

The graph edit operations are performed on pattern-region $G_{g(i)}^{pr}$ and its edge-bundles $R_{m\leftrightarrow pr}^i$ to match them against selected source-region $G_{g(i)}^{sr}$ and its connector-edges $R_{m\leftrightarrow sr}^i$.
Example: *incremental* graph-pattern matching (*phase 2*)

Query graph

Pattern graph

Matched graph

Input graph

*Match*

\[
G_2^m = G_1^m + (R_2^m \leftrightarrow \text{pr} \oplus G_2^\text{pr})
\]

\[
G_2^m = G_1^m + (R_2^m \leftrightarrow \text{sr} \oplus G_2^\text{sr})
\]

Kamran Sartipi: ICSM 2003, Book LAMBERT Academic
Objective: generating highly cohesive modules

Internal-edge deletion cost must relate to:
- **M**: maximal similarity between two nodes in the region
- **s**: similarity between corresponding nodes
- **k**: number of already matched nodes in the module
- **d**: number of deleted edges between two nodes

\[
c = \frac{M - s}{k} + \frac{0.25 d}{s} \frac{d}{k}
\]

Two cases: matching nodes 15 and 7
**Imported & exported connector-edge deletion costs**

**IMPORT**

1. “r” = number of remaining edge-bundles including the current edge-bundle
2. Keep “r” edges from the current edge-bundle and delete the rest with cost “zero”
3. Match the edges from “r” edges in edge-bundle
4. From “r” edges, each edge that is not matched, is deleted with cost:

**EXPORT**

IF one or more edges matched from edge-bundle THEN delete unmatched edges with cost “zero”
ELSE delete all edges with cost: \(0.25 \times C_{in}^{ed}\)

---

**Example: r = 3, and 2 edges matched**

- Deleted edge
- Matched edge
- Matched edge
- Matched edge

**Example: 2 edges matched**

- Deleted edge
- Matched edge
- Matched edge
- Matched edge

Cost = zero
Generating pattern-graph from query-graph

Query-graphs with 2 nodes

Imported edge-bundles

Exported edge-bundles

Query-graph with 4 query-nodes

Generated pattern-graph at phase 4
**Edge matching for imported edge-bundles**

- **Part of pattern-graph at phase i**
- **Three edges matched. Cost = max**
- **No edge matched. Redirect with cost**
- **One edge matched. Others deleted with some cost**
- **Exceeds max edges**
- **Duplicate import**
- **Duplicate import is not counted. Cost = 0**
- **No edge matched. Edge-bundle deleted with cost. Min # edges may be violated**

---

*Kamran Sartipi: ICSM 2003, Book LAMBERT Academic*
Edge matching for exported edge-bundle

Part of pattern-graph at phase i

Three edges matched
Cost = 0

No edge-bundle deleted
Cost = max

One edge matched.
Cost = 0

Edge-bundle redirected with cost.

matched node

No edges matched.
Edges deleted.
Min # may be violated.

Kamran Sartipi: ICSM 2003, Book LAMBERT Academic
**Steps for incremental pattern generation**

1) Select main-seed for next module using tool provided techniques.

2) Recover next module with no link constraints

3) Based on the interaction with other components, and user’s objectives define the constrained links for this module.
   * Maximum range is used to encourage high interaction
   * Minimum range is used to restrict the number of interaction

---

**Architectural pattern using AQL query**

**Recovered architecture**

---

*Kamran Sartipi: ICSM 2003, Book LAMBERT Academic*
Techniques to Address Tractability

- Incremental recovery by dividing the search space into sub-spaces

- Hierarchical recovery
  - Decomposing system into subsystem of files
  - Decomposing a subsystem into modules of F/T/V

- Sub-optimal search techniques, e.g., bounded path-queue A* (BQ-A*)

- Implementation techniques
A* search with Bounded path-queue

Sub-optimal solution to achieve tractable search.

- A* produces queue of sorted incomplete paths.
- Storing, sorting, duplicate path checking are bottlenecks.
- In successful search most of paths at the end of queue are not expanded.
- Max / min thresholds: multiples of the size of domains.
Space Complexity Reduction

Number of stored paths in path-queue

Under-estimate cost

Cost estimation for one node-matching

Bounded path-queue min/max sizes

Number of placeholders

Kamran Sartipi: ICSM 2003, Book LAMBERT Academic
User assistance

- Statistical Metrics
  - Overall association among files
  - Fan-in fan-out
  - Design views

- Visualization
  - Simplifying the graph views
  - Browsing mechanism through HTML pages

- Assistance with pattern generation
  - Identifying the locus of interactions
Representing the architecture using graph visualizer (Rigi)

- Different types of links between boxes:
  - Association-links
  - Entity-usage links
- Association-links with different strengths to simplify the view
- Viewing the locus of interaction among entities to evaluate the recovery process
- Insight into the system before starting the recovery
- Manual recovery

Kamran Sartipi: ICSM 2003, Book LAMBERT Academic
Architecture of Apache 1.2.4

Partitioning
Representing the architecture using Web browser (NetScape)

- Hypertext links to actual entities in the source file.

- Information presented includes:
  - Evaluation metrics: modularity quality, average similarity
  - Statistical information for link-constraint violations
  - Interactions among components
  - Browsing the query
  - Switch between file-level and function-level analysis

```
 File-level Analysis: XFIG
```

```
Violated Link-Constraints
- SS: MAX of the EXPORTED link/IN violated 49 times.

Modularity (0.0027)...Overall avg-similarity (0.113)

SubSystem: S1-S4 avg-similarity (0.249)
```

```c
Top S1-S4 S3 S1 S4 Peer
```

```c
Imports:
- Resources:
  1. From S3 (79/165)
  2. From S2 (17)
```

```c
Exports:
- Resources:
  1. To S3 (79/247)
  2. To S2 (22)
  3. To S5 (1)
```

```c
Contains:
- Files:
  1. C:\\-54 \_v\_plastic\(;\) (32) (0.629)
  2. C:\\-54 \_v\_id1\(;\) (0.679)
  3. C:\\-28 \_v\_plastic\(;\) (3.04)
  4. C:\\-53 \_v\_drum\(;\) (0.316)
  5. C:\\-55 \_v\_test\(;\) (0.054)
  6. C:\\-53 \_v\_search\(;\) (0.268)
  7. C:\\-13 \_o\_box\(;\) (0.226)
  8. C:\\-55 \_b\_sphere\(;\) (0.164)
  9. C:\\-55 \_is\_con\(;\) (0.189)
  10. C:\\-55 \_is\_con\(;\) (0.167)
  11. C:\\-11 \_t\_bbox\(;\) (0.021)
  12. C:\\-55 \_t\_bbox\(;\) (0.216)
  13. C:\\-55 \_t\_bbox\(;\) (0.167)
  14. C:\\-55 \_t\_bbox\(;\) (0.239)
  15. C:\\-55 \_t\_bbox\(;\) (0.165)
  16. C:\\-55 \_t\_bbox\(;\) (0.144)
  17. C:\\-55 \_t\_bbox\(;\) (0.044)
```

```c
```
Validation of the recovery

- Modularity quality
  - Connectivity based
  - Association based
- User investigation of the graphs
  - Simplified graphs
- Conformance with documented architecture
  - Precision and Recall
Accuracy of the recovered architecture

- Clips expert system
  - 40 KLOC
  - 44 files
  - 736 functions
  - 161 global vars
  - 54 aggregate types

- Xfig drawing editor
  - 74 KLOC
  - 98 files
  - 1662 functions
  - 1356 global vars
  - 37 aggregate types

### Recovered subsystems

<table>
<thead>
<tr>
<th>Recovered subsystems</th>
<th>No. of files</th>
<th>Clips subsystems</th>
<th>No. of files</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>11</td>
<td>- Defrule structures</td>
<td>13</td>
<td>82%</td>
<td>70%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Inference engine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
<td>- Rule manipulation</td>
<td>6</td>
<td>50%</td>
<td>83%</td>
</tr>
<tr>
<td>S3</td>
<td>4</td>
<td>- Object</td>
<td>3</td>
<td>75%</td>
<td>100%</td>
</tr>
<tr>
<td>S4</td>
<td>4</td>
<td>- Expression eval</td>
<td>4</td>
<td>75%</td>
<td>75%</td>
</tr>
<tr>
<td>S5</td>
<td>10</td>
<td>- System function</td>
<td>7</td>
<td>49%</td>
<td>57%</td>
</tr>
<tr>
<td>rest-of-sys</td>
<td>5</td>
<td>User interface</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recovered subsystems</th>
<th>No. of files</th>
<th>Xfig subsystems</th>
<th>No. of files</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1-S4</td>
<td>37</td>
<td>editing &amp; utility &amp; drawing</td>
<td>47</td>
<td>81%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X-windowing</td>
<td>28</td>
<td>78%</td>
<td>64%</td>
</tr>
<tr>
<td>S3</td>
<td>20</td>
<td>editing &amp; utility</td>
<td>37</td>
<td>65%</td>
<td>31%</td>
</tr>
<tr>
<td>S5</td>
<td>10</td>
<td>file manipulation</td>
<td>16</td>
<td>70%</td>
<td>44%</td>
</tr>
<tr>
<td>rest-of-sys</td>
<td>8</td>
<td>5 zero size files</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Presented an interactive environment for architectural recovery and evaluation, and the supporting toolkit

- Highlights of the approach:
  - Modeled the recovery process as “graph pattern matching”
  - Used data mining techniques to define similarity metric
  - Limited the complexity of recovery process by two techniques
  - Developed a query language based on ADL features
  - Represented the recovery result through HTML pages and graphs to be visualized
On Modeling Software Architecture Recovery as Graph Matching

Kamran Sartipi

Department of Computing and Software
McMaster University
Canada
Sartipi@mcmaster.ca

IEEE International Conference on Software Maintenance (ICSM 2003)