A Query-based Approach to Software Architecture Recovery

Kamran Sartipi
Kostas Kontogiannis
University of Waterloo
{ksartipi, kostas}@swen.uwaterloo.ca

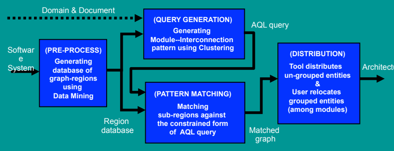
Software Architecture Recovery

Definition: Extracting high-level information from some low-level software representation such as source code

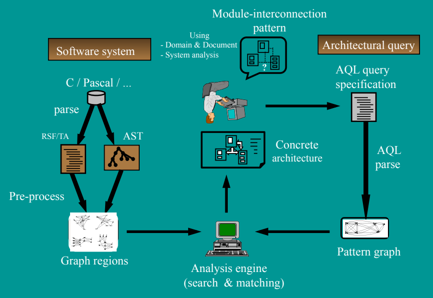
Approach

The user specifies a high level abstraction of the system as a graph of modules and interconnections, where each module (one node of graph) represents a group of placeholders for the system entities (i.e., func, type, var) to be instantiated, and each bundle-of-interconnections between two modules (one edge of graph) represents data/control-dependencies. The min/max sizes and the types of placeholders / interconnections are free parameters to be decided by the user.
This abstract graph (defined using our Architecture Query Language, AQL) is then expanded into a pattern graph for a part or whole system architecture to be recovered. The architectural recovery process tries to find a series of graph edit operations (i.e., node/edge insertion/deletion) with minimum cost that if applied on the pattern graph, the result would match with a sub-graph of the system graph of entities and relationships. The method is known as inexact graph matching.
A branch and bound search algorithm with association-based score function is used for the matching process.The system is first decomposed into a group of subsystems of files, then each subsystem can be decomposed into a number of modules of functions, aggregate types and global variables.

Framework for Architectural Recovery

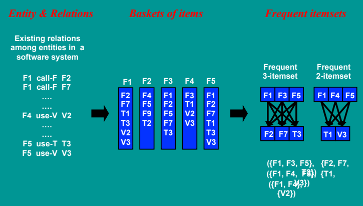

Domain & Document
(QUERY GENERATION)
Generating Module--Interconnection pattern using Clustering
AQL query
Software System
(PRE-PROCESS)
Generating database of graph-regions using Data Mining
(DISTRIBUTION)
Tool distributes un-grouped entities & User relocates grouped entities (among modules)
Architecture
(PATTERN MATCHING)
Matching sub-regions against the constrained form of AQL query
Region database
Matched graph

Scenario for Architectural Recovery Based on Pattern Matching



Association using Data Mining Technique (Apriori)

- Discovery of Interesting and non-trivial relations among data in large databases.
- Frequent itemsets: a collection of items that all exist in each basket of a group of baskets



Software System as Attributed Relational Graph

An "ARG" is a six-tuple G = (N, R, A, E, f, g):

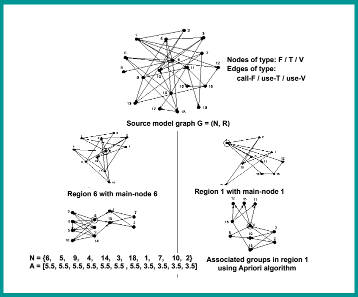
- N = {n_1, n_2, …, n_n}: attributed vertices (nodes)
- R = {r_1, r_2, …, r_m}: directed attributed edges (relations)
- A & E: alphabets for node & edge attributes
- f & g: node & edge labeling functions

Example of attributes in software system:

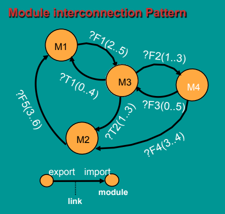
- Label: a unique string for nodes only
- Type: identifiers to classify nodes and edges
- Location: two integers for *file#* and *line#*

$f(n_5)$ = ("/u/…/foo", F, 6, 47)
$g(r_{28})$ = ((n_5, n_{34})) = (call-F, 6, 92)

Decomposition of System Graph into Regions


Nodes of type: F / T / V
Edges of type: call-F / use-T / use-V
Source model graph G = (N, R)
Region 6 with main-node 6
Region 1 with main-node 1
N = {6, 5, 9, 4, 14, 3, 18, 1, 7, 10, 2}
A = [5.5, 5.5, 5.5, 5.5, 5.5, 5.5, 5.5, 3.5, 3.5, 3.5, 3.5]
Associated groups in region 1 using Apriori algorithm

Architecture Query Language (AQL)



```
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-MODULE
```

- A query is modeled as a multi-graph of nodes and edges
- Each node represents an abstract module to be instantiated with system entities.
- Each edge represents a group of link-constraints between two modules in the form of import / export of resources (func / type / var).
- Each module has one (or more) main-seeds which determine the domain of entities to be put in the module, and zero or more seeds which specialize the query.

Graph Matching

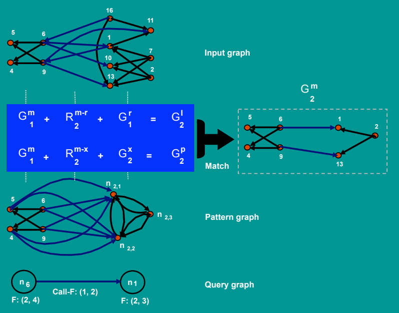
- *f: G → G'* maps the nodes and relations of graph G onto graph G' :
- Exact graph matching:
  - Exact set of nodes and edges of G that is isomorphic to G'
- Inexact graph matching:
  - An optimal sequence of graph edit operations, such as: insertion / deletion / relabeling of nodes and edges of G so that G and G' become isomorphic

Graph Matching Process

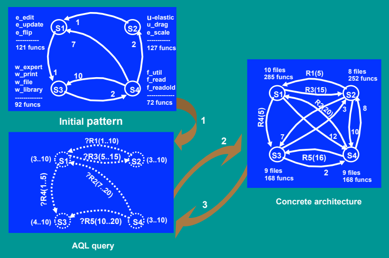
$$\left.\begin{array}{l} G^m_{i-1} + R^{m-r}_i + G^r_{g_i} = G^l_i \\ G^m_{i-1} + R^{m-x}_i + G^x_i = G^p_i \end{array}\right\} \xrightarrow[\text{Match}]{} G^m_i$$

- At each phase *i* of the matching process, incremental input graph G^l_i is inexactly matched against incremental pattern graph G^p_i which results in incremental matched graph G^m_i
- We perform graph edit operations on:
  expanded graph G^x_i and its glue edges R_i^{m-x}
  to match them with
  selected region G^r_{gi} and its glue edges R_i^{m-r}

Graph Matching Example



Experiment with Xfig 75 KLOC: Pattern … Query … Architecture



Result Xfig: Netscape browser and Rigi graph visualizer