Software Architecture Recovery Using Data Mining Techniques

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

Approach

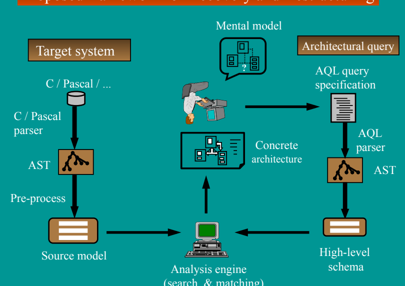
We propose a framework for software architecture recovery and restructuring. In this framework, the user specifies a high level abstraction of the system using a structural pattern language (we call it Architecture Query Language, AQL). Then, a pattern matching engine provides an optimal match between the given pattern and a decomposition of the legacy system entities into modules while satisfying the inter/intra-module constraints defined by the pattern.
The data mining technique Apriori is used to limit the search space.
A branch and bound search algorithm models the constraints in the pattern as a Valued Constraint Satisfaction Problem. The decomposition is performed both at the system level (groups of files) and subsystem level (groups of func / type / var).

Software Architecture Recovery

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

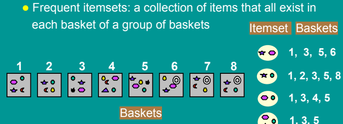
- Constitutes a major task in software maintenance
- Should relate to specific re-engineering requirements
- Major approaches:
  - Clustering techniques based on static and dynamic properties.
  - Constraint satisfaction to satisfy user-defined constraints.
  - Query-based techniques based on specialized queries and high-level architectural styles

Proposed framework for Recovery and Restructuring



Data Mining Technique (Apriori)

- Discovery of Interesting and non-trivial relations among data in large databases.
- Apriori: a fundamental data mining algorithm [Agrawal]
  - Frequent itemsets: a collection of items that all exist in each basket of a group of baskets

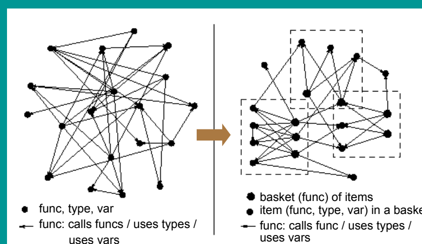


- Discovery of association rules (X --> Y)
  e.g., 60% of the transactions that contain the set also contain the set

  i.e., --> with the confidence level of 60%

Application of Data Mining in Recovery Process

- Data mining allows to reveal cohesive groups of entities in the form of bi-partite sub-graphs in the graph of the target system.
- The bi-partite sub-graphs are the basis for extracting the source model for the matching process.
- The source model is a forest of trees, and each tree consists of all entities that are associated with an individual node in the bi-partite sub-graph.
- Each tree contains all possible entities that can be put in a module, we call it the domain of a module. The root of the tree is called a main-seed.

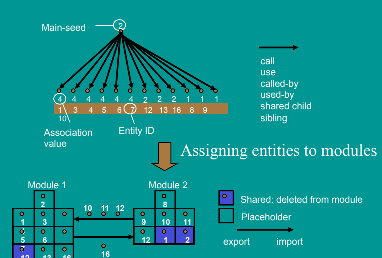


(a) Un-processed graph of the target system

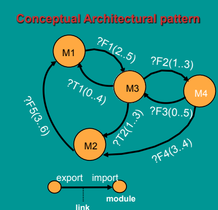
(b) Bi-partite sub-graphs resulted from applying Apriori algorithm on graph (a)

Model of the matching process

- A domain-selection algorithm performs an exhaustive search to find the best candidate domains for the modules in the query.
- The criteria for domain-selection include:
  - High average of the association values between each entity in a domain and the corresponding main-seed,
  - Low level of scattering of the domain entities into the system files, and
  - Large domain size
- The matching process selects the entities for each module based on high association value and high average clustering value to the group of entities already selected for the module.
- Each allocation of an entity to a module must satisfy both the similarity-constraints (i.e., association and clustering values) and the link-constraints (i.e., abstract links between modules in the query).
- Import s/ Exports are manifestation of link-constraints between modules.



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-ENTITY
```

- 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.

User interfaces

- Web browser (Netscape):
  - Hypertext links to actual entities in the source files.
  - Various information: distribution of recovered entities into files, browsing the query, statistical data for link-constraint violation, links between modules.
- Graph visualizer (RIGI): property of recovered entities (bi-partite sub-graphs).

Case study: CLIPS, 40 KLOC in C

- The graph of a recovered module with 18 functions using RIGI graph visualizer
- A query and its solution represented using Netscape browser