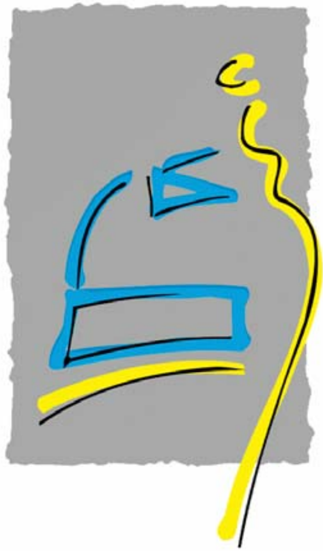
FZI Forschungszentrum Informatik
an der Universität Karlsruhe

Combining Clustering with Pattern Matching for Architecture Recovery of OO Systems

Mircea Trifu
Markus Bauer



WSR 2004, Bad Honnef

Overview

- Context and Problem
- Related Work
- Our Approach
  - Gathering Architectural Clues
  - Adapting Dependecy Measures
  - Clustering
- Evaluation
- Future Work
- Summary

Context and Problem

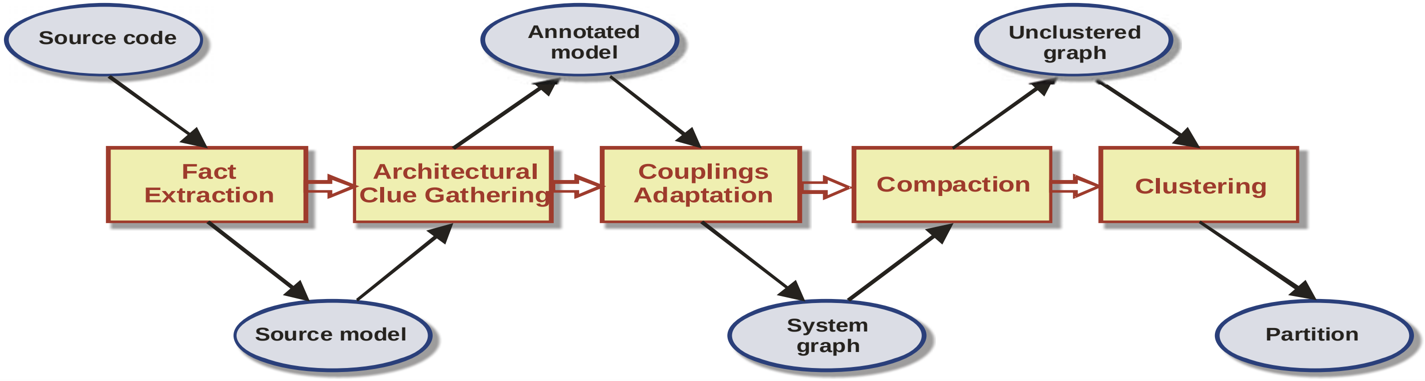
- Software systems age over time
  - Structures erode, knowledge about the system fades
  - Evolution of systems becomes difficult and expensive
- Problem: Recover a system‘s architecture
  - to achieve a better understanding of the system
  - to identify spots where the structure needs improvement
- Solution: Develop methods and tools that automate the task of architecture extraction

Related Work

- Pattern based approaches
  - identify structures by graph- or pattern matching techniques
  - detect structural problems [Ciupke2001], design patterns [Prechelt1996, Antoniol1998], user defined architectural structures [Sartipi2001]
  - mainly recognize „micro structures“ (method or class level)
  - do not cover quality properties for the subsystems (coupling, cohesion,…)
- Approaches based on clustering
  - group system‘s entities based on their syntactic dependencies
  - used mainly for reverse engineering systems written in procedural [Mancoridis1999, Koschke2000] and OO languages [Rayside2000], [Trifu, Bauer2001], [eAbreu2000]
  - neglect the role the system‘s entities play in the architecture
  - often produce system decompositions that are of not much meaning to developers

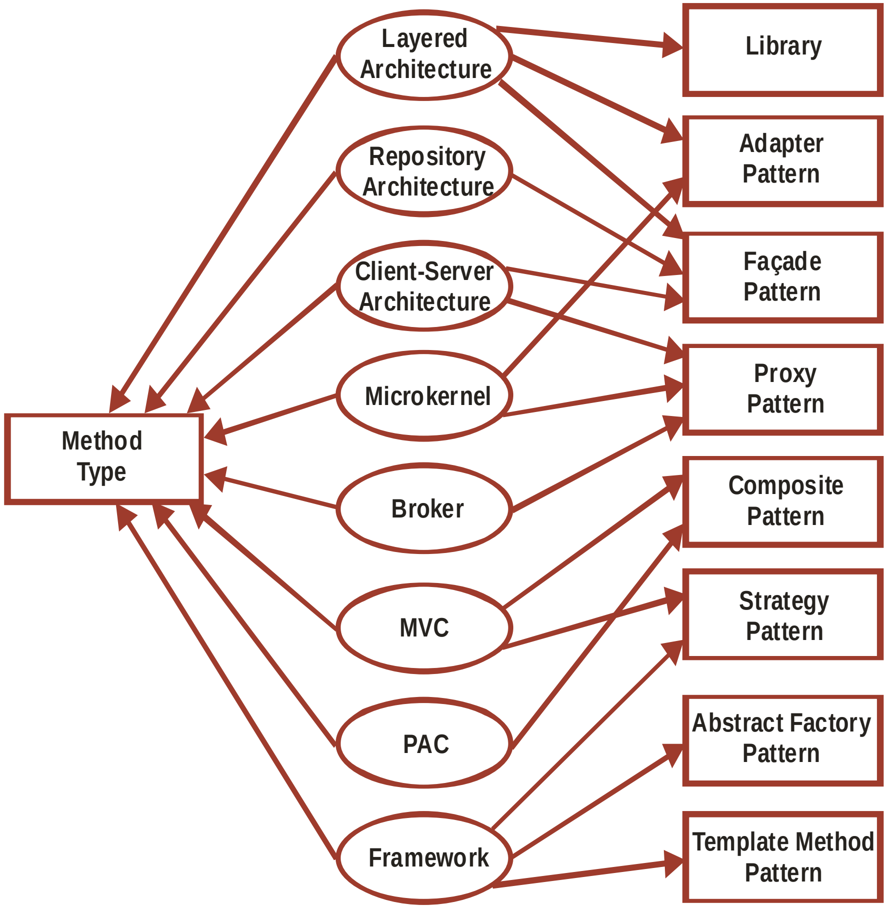
Our Approach

- Combine pattern based approaches and clustering → Pattern based, adaptive clustering
- Pattern matching
  - collects hints about the role syntactic elements and their relationships play in the system‘s architecture
- Cluster analysis
  - groups elements into subsystem candidates based on relationships
  - makes use of these hints



Pattern Matching

- Exploit architectural patterns
  - Architectures employ patterns
  - Detection of architectural patterns is difficult (structures erode!)
  - Architectural patterns use fine grained patterns (fingerprints, clues), those are easier to detect
  - Fingerprints have predefined roles
  - Roles provide a means to rate dependencies

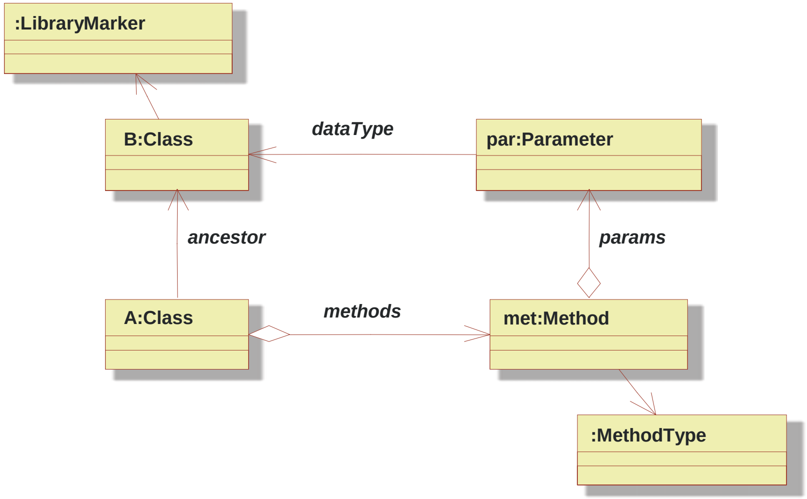


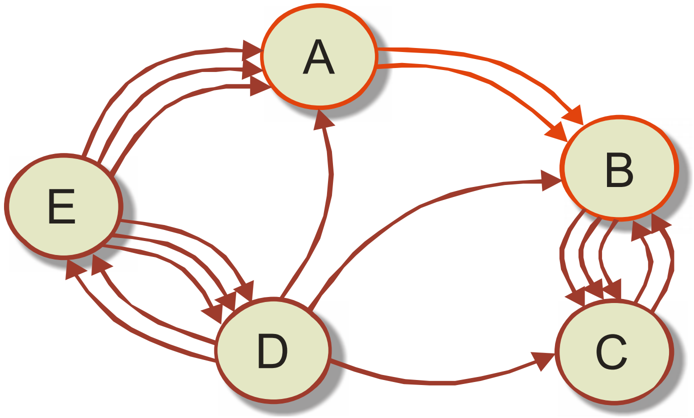
Architectural clues

- Architectural clues can be detected automatically
  - Classification of methods
    - What role does a method have? (delegation, accessor, ...)
    - What statute does it have (wrt. inheritance)? (new, (re-)implementation, extension, …)
    - How is it used? (initializer, interface, implementation, ...)
  - Detection of library code
    - Usage count on the interface
  - Detection of design patterns (GoF)
    - Adapter, Facade, Proxy, Composite, Strategy, Abstract Factory, Template Method
- Result: annotated structural model

Construction of the System Graph

- Source code model
  → Weighted (multi-)graph
  - Classes = nodes
  - Dependencies = edges
    - Inheritance
    - Aggregation
    - Association
    - Variable accesses
    - Method calls
    - Indirect coupling
- Weights are influenced by the detected clues (according to their standard roles)

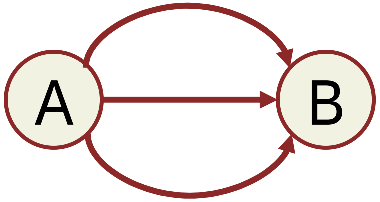




Examples: Calls, Indirect Coupling

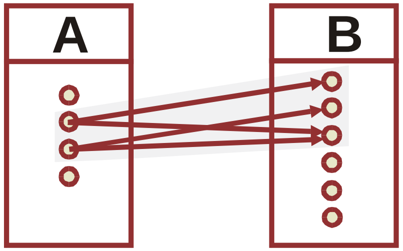
Calls

- Calls between classes A und B



Context	Weight
Library	**0.5**
Standard	**1**
Composite	**5**

- Adjust weights for the calls (according to the clues detected)

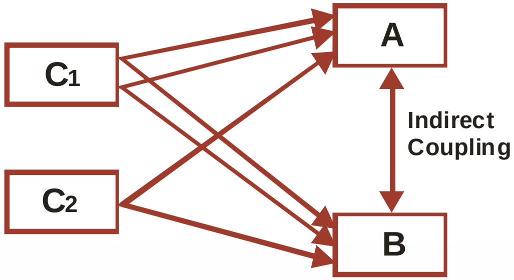


- Use metrics to aggregate the information about calls between A and B

Indirect coupling

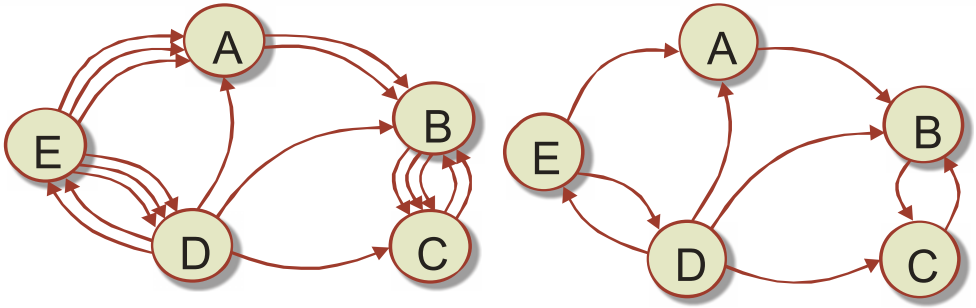
- Elements that are frequently used together belong together

$$IndCoupling(A,B) = \sum_{C} \frac{noMethods}{methods(C)}$$



Compaction and Clustering

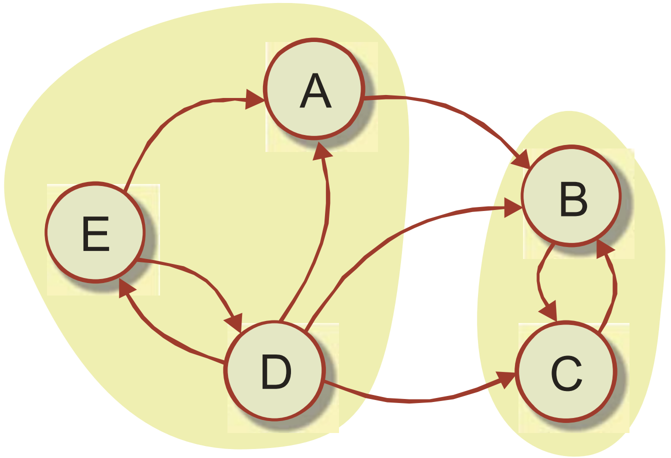
- Compaction:
  - Transform the multi-graph into a standard graph



$$dsim(A,B) = \sum_{i=1}^{7} w_i \cdot c_i$$

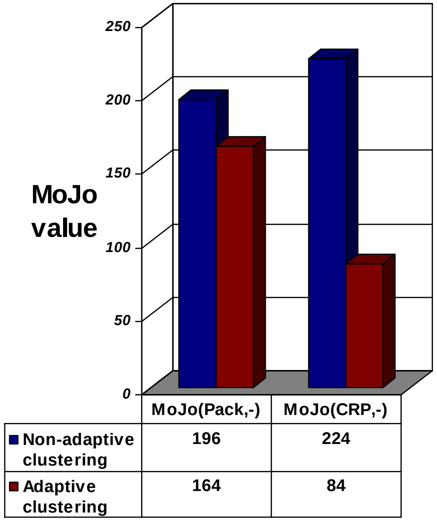
$$sim(A,B) = \max(dsim(A,B), dsim(B,A))$$

- Clustering:
  - Employ mature standard algorithms
  - Goal: Group the nodes of the graph
  - Right now: a modified MST algorithm

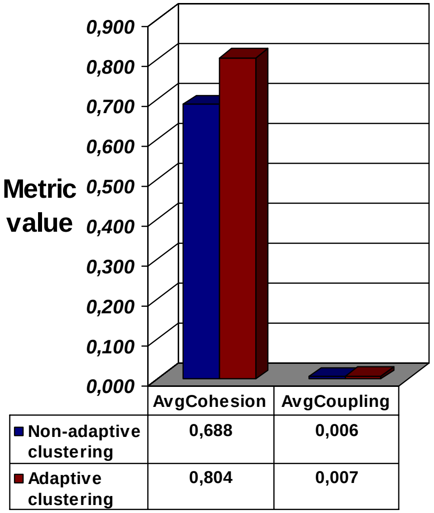


Evaluation

- ACT: Tool-Prototype in Java
- Comparing traditional vs. adaptive clustering using Java AWT as case study
  - Package structure and CRP structure vs. clustering
  - Cohesion and coupling properties



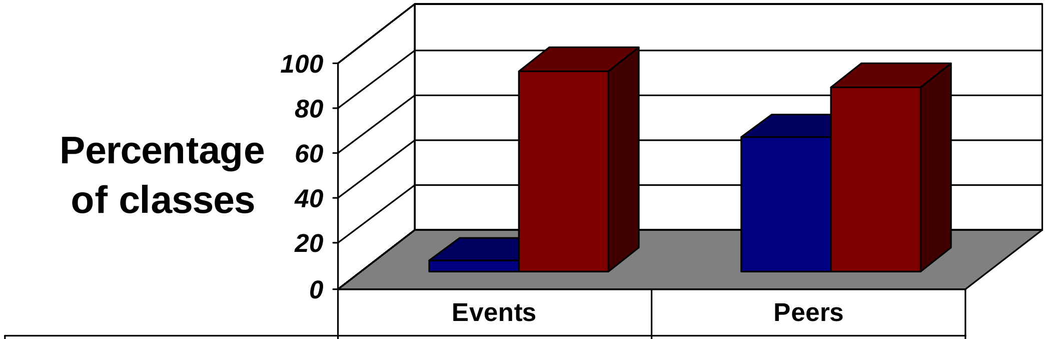
	MoJo(Pack,-)	MoJo(CRP,-)
🟦 Non-adaptive clustering	196	224
🟥 Adaptive clustering	164	84



	AvgCohesion	AvgCoupling
🟦 Non-adaptive clustering	0,688	0,006
🟥 Adaptive clustering	0,804	0,007

Some Details...

- ► Semantically related entities have been grouped together:
  - ● `Menu, MenuItem, MenuContainer, MenuShortcut`
  - ● `TextComponent, TextArea, TextField`
- ► Successful separation of classes from different abstraction levels and with different roles



	Events	Peers
■ Non-adaptive clustering	4,54	59,25
■ Adaptive clustering	88,63	81,48

- ► Comparable results for 2nd casestudy: SSHTools

Future Work

- Consider other types of syntactic interactions
  - Cast expressions
- Identify additional clues
  - Observer pattern; CORBA, COM calls, ...
- Experiment with different clustering algorithms
- Experiment with more case studies
  - Perform a more detailed comparison with other approaches
  - Collect more evidence about clue usage
  - Tune the thresholds and weight values
- Integrate the technique in our software assessment tool suite

Summary

Our work contributes:

- ► A new approach for architecture extraction
  - combining the strengths of pattern based and clustering approaches
  - evaluating fingerprints of architecture information
- ► Useful metrics to express dependencies
  - Call metrics, indirect coupling
- ► A powerful way to „correctly“ cluster:
  - framework-application settings
  - layered architectures
  - library code

Questions and Comments