Feature Matching in Model-Based Software Engineering

Alar Raabe

Tallinn Technical University

Contents

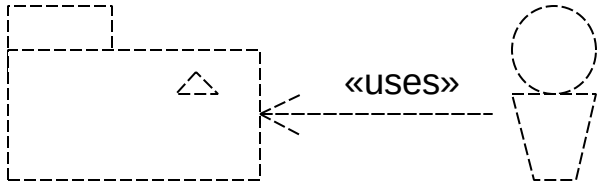
- Introduction
- Usage of models in software engineering
- Domain analysis
- Feature matching
- Related work
- Conclusions

Introduction

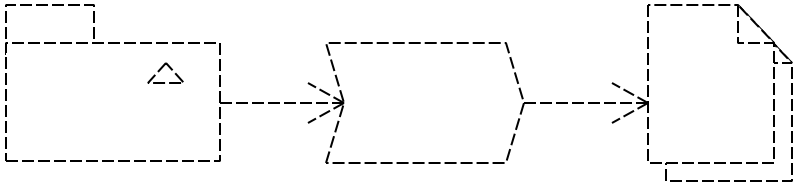
- Today's business
  - More dependent on software
  - Constantly changing
- Requirements for business information systems
  - Rapid delivery of initial results
  - Effortless change during the life-cycle
  - Independence of business know-how from information technology know-how
  - Minimal cost (acquisition and ownership)
- Context of given research
  - Insurance software product-line architecture, tools and method for producing product-line members

Usage of models in software engineering

- For documentation
  - Analysis
  - Design
  - Implementation
  - …



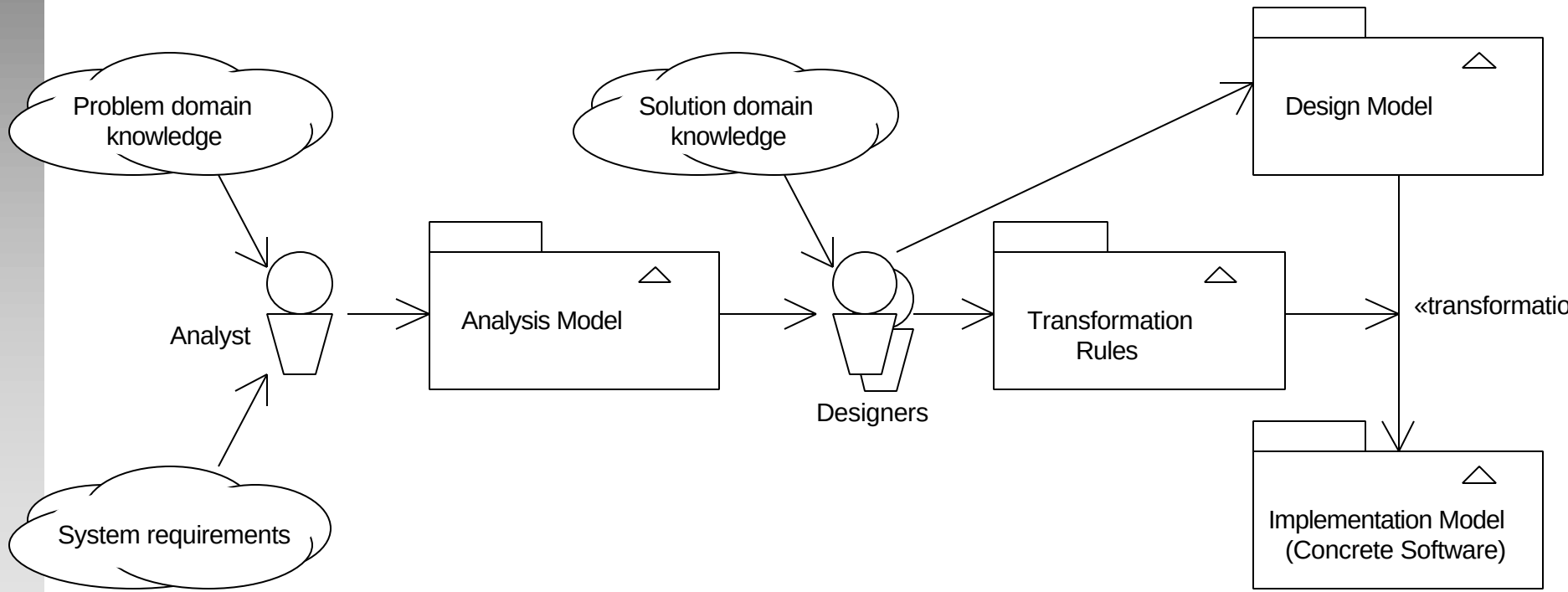
- As source artefacts (in model-based methods)
  - Results of
    - analysis – problem statement
    - design | implementation – specification of solution
  - Sources for
    - compilation | generation
    - interpretation | execution



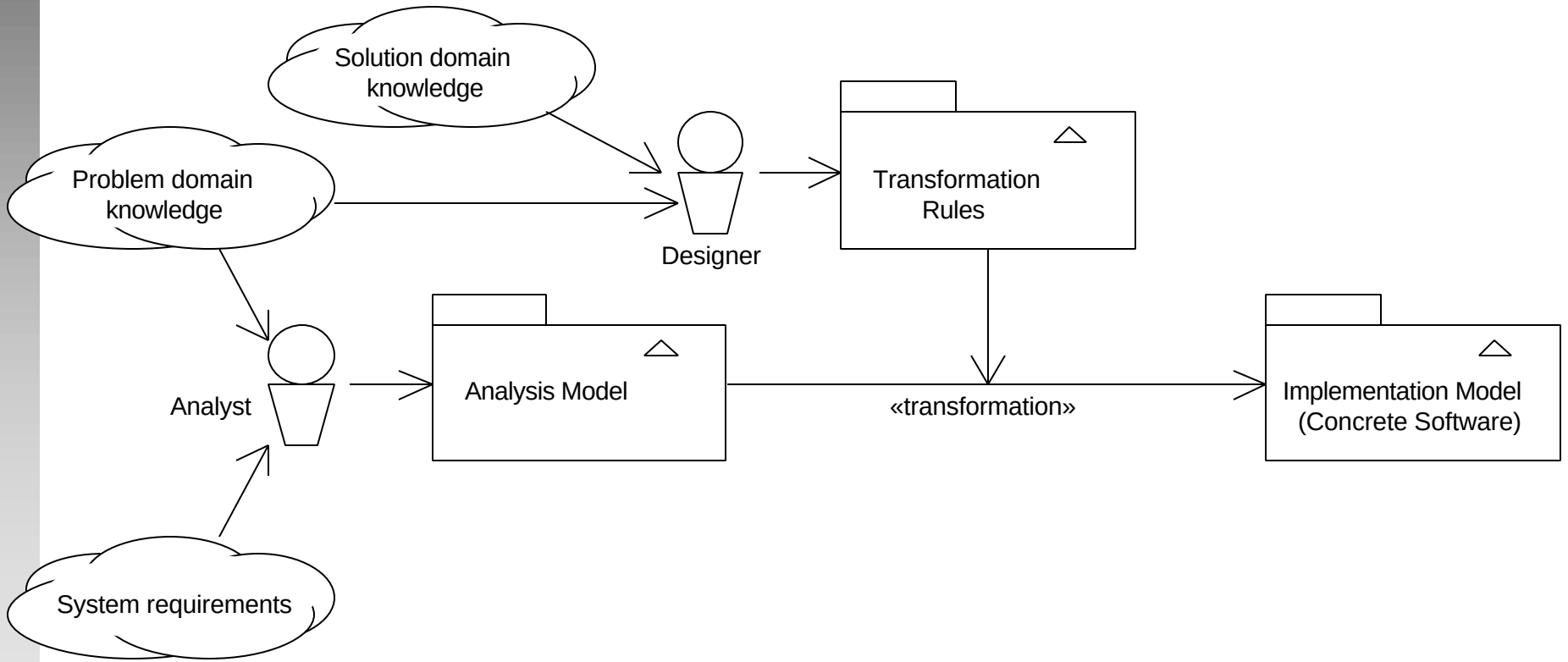
Model-based software engineering methods

- Methods where models are main artefacts
  (some, or all other artefacts are derived from them)
- Model-based approaches for
  - Real-time and embedded systems
    - Model-Integrated Computing (MIC) and model-based software synthesis – (Vanderbilt Univ. (ISIS), 1993; Abbott et al., 1994)
    - Model-based development – (Mellor, 1995)
  - Generative programming
    - GenVoca – (Batory, 1992)
    - Family-Oriented Abstraction, Specification, and Translation (FAST) – (Weiss, 1996; AT&T, Lucent, 1999)
  - Software system families (a.k.a. product-lines)
    - Model-Based Software Engineering (MBSE) – (SEI, 1993)
  - Integration and interoperability
    - Model-Driven Architecture (MDA) – (OMG, 2001)

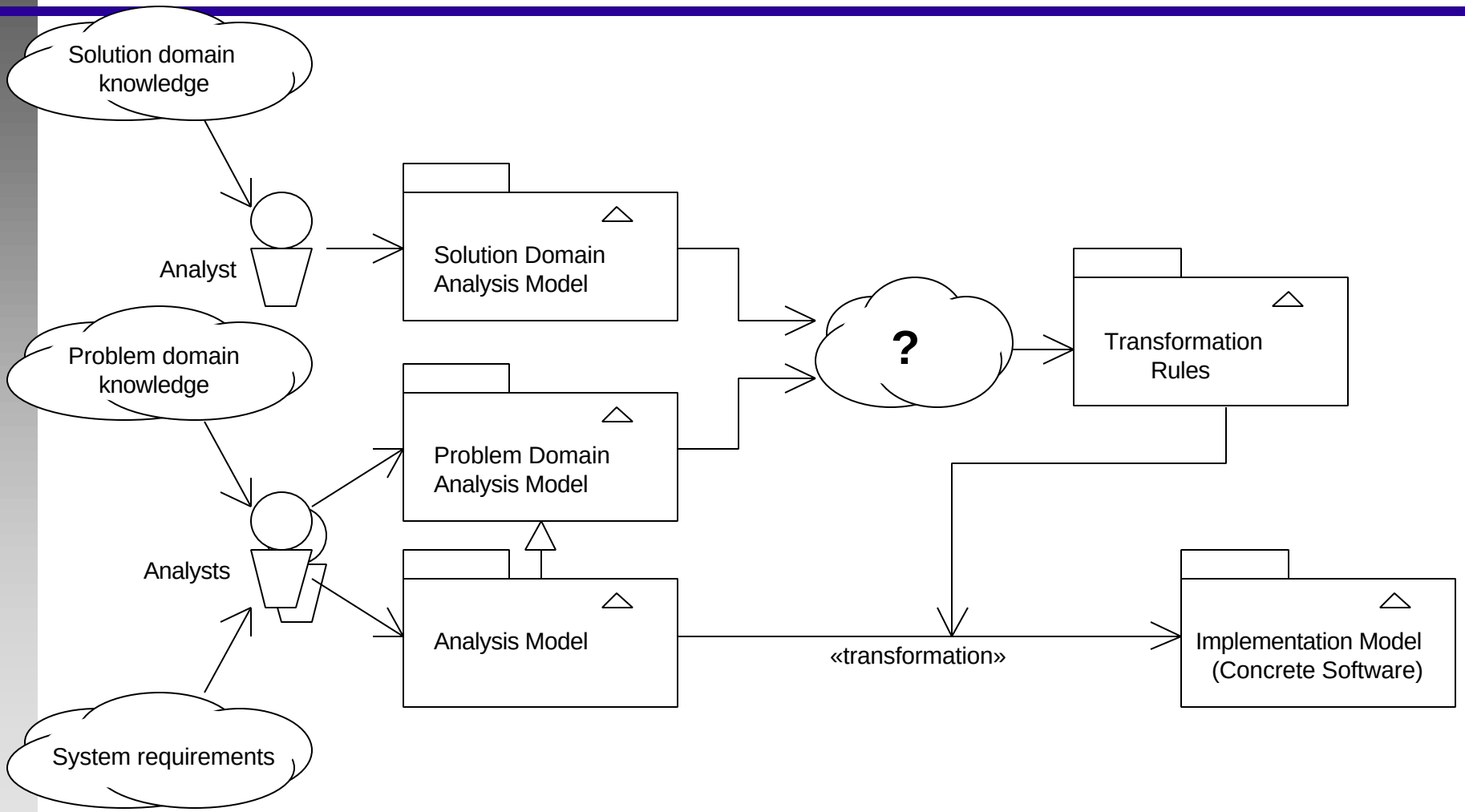
Traditional MBSE approach (1)



Traditional MBSE approach (2)



Proposed MBSE approach



Domain analysis

- Domain
  - an area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area (UML)
- Domain Analysis
  - Domain scoping – select and define domain of focus (context)
  - Domain modelling – collect the relevant domain information and integrate it into a coherent domain model
- Domain model
  - A body of knowledge in a given domain represented in a given modelling language
    - Scope (boundary conditions of the domain)
    - Domain knowledge (elements that constitute the domain)
    - Generic and specific features of elements and configurations
    - Functionality and behaviour

Domain analysis methods

- Domain analysis methods
  - Language based
  - Algebraic (formal)
  - Object-oriented
  - Aspect-oriented
  - Feature-oriented
  - Combined approaches → feature-oriented + …
- Domain analysis methods based on features
  - Feature-Oriented Domain Analysis (FODA) – SEI
  - Feature-Oriented Reuse Method (FORM) – K. Kang
  - Domain Engineering Method for Reusable Algorithmic Libraries (DEMRAL) – Czarnecki, Eisenecker
  - …

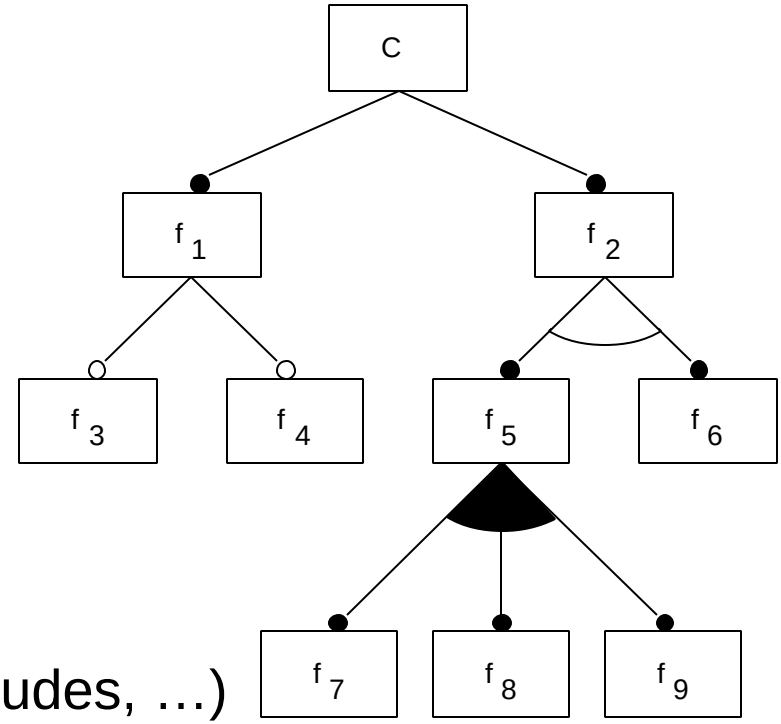
Feature modelling

- Feature modelling (a.k.a feature analysis)
  - is the activity of modelling the common and the variable properties of concepts and their interdependencies

- In feature modelling
  - ***Concepts*** are any elements and structures of the domain of interest
  - ***Features*** are qualitative properties of concepts
  - ***Feature model*** represents the common and variable features of concept instances and the dependencies between the variable features
  - Feature model consists of a ***feature diagram*** and additional information

Feature diagram

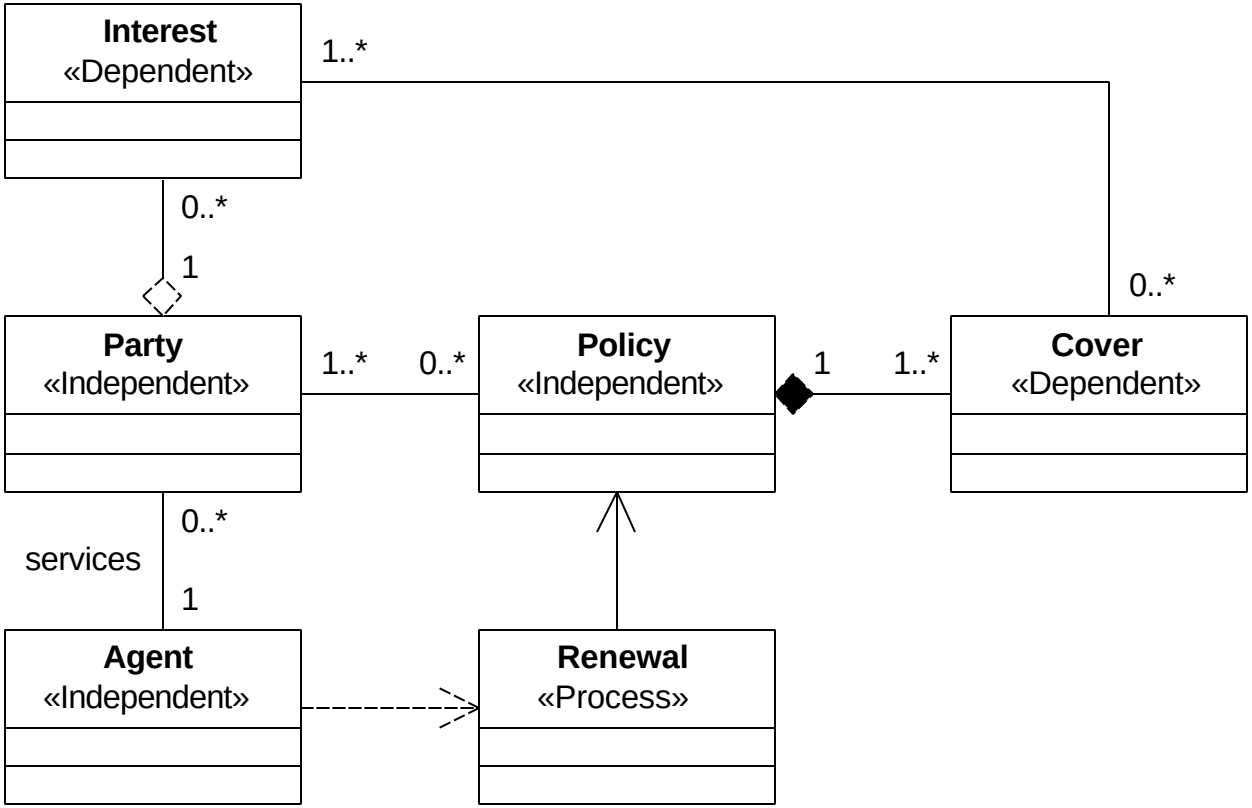
- Tree-like diagram where
  - The root node represents a concept, and
  - Other nodes represent features
- Feature types
  - Mandatory features (f_1, f_2, f_5, f_6)
  - Optional features (f_3, f_4)
  - Alternative features (f_5, f_6)
  - Or-features (f_7, f_8, f_9)
- Constraints between features
  - Composition rules (requires, excludes, …)



Feature types

- FODA feature types
  - Context features – performance, synchronization, …
  - Operational features – application functions
  - Representation features – visualization, externalization, …
- FORM feature types
  - Capabilities
  - Operating environment
  - Domain technologies
  - Implementation techniques (domain independent)

Example problem domain model (Insurance)

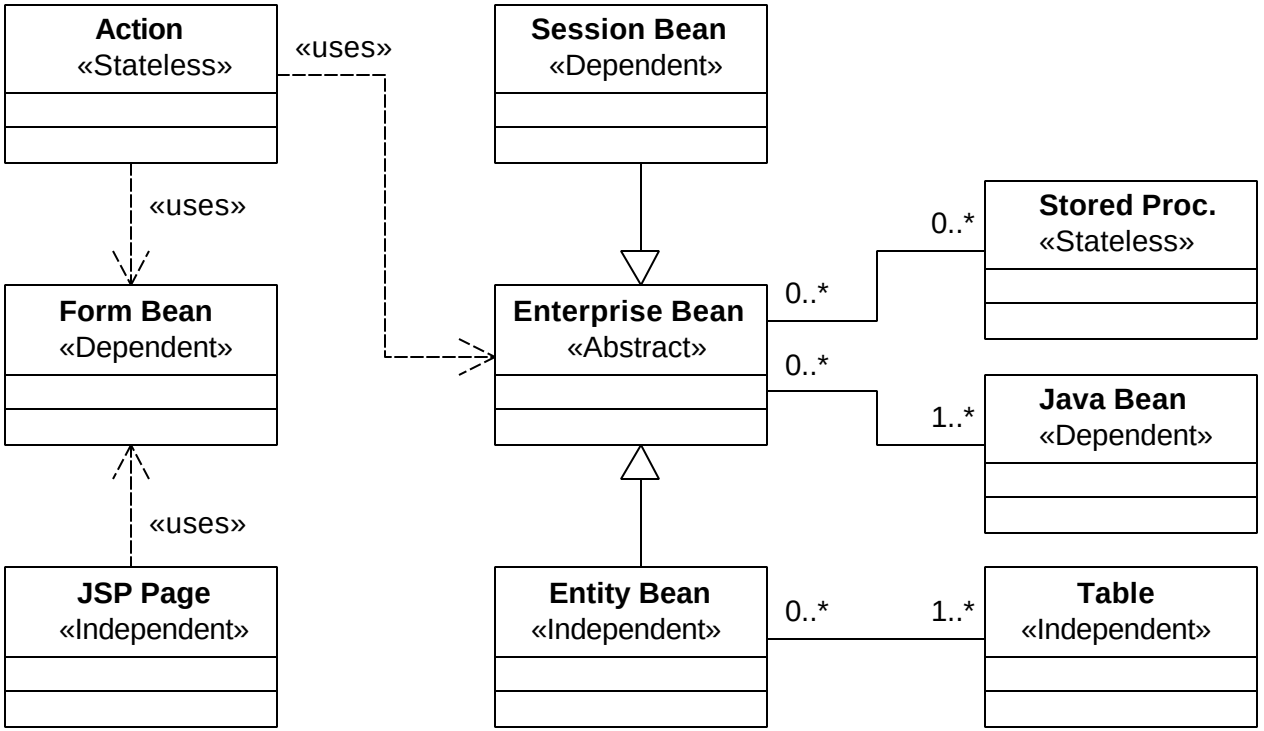


Example problem domain model – features independent of domain

- Concept “Policy” – independent business object
  - Features (domain independent)
    - Has identity
    - Independent
    - Has state
    - Persistent → Storable, Searchable
    - Viewable → Modifiable

- Concept “Renewal” – business process
  - Features (domain independent)
    - No identity
    - No state
    - Transient
    - Business behavior → Asynchronous

Example solution domain model (J2EE + Struts + RDB)



Example solution domain model – features independent of domain

- Concept “Entity Bean”
  - Features (independent of domain)
    - Identity
    - State
    - Persistent → Storable, Searchable
    - Behavior
- Concept “Session Bean”
  - Features (independent of domain)
    - No identity
    - State is optional
    - Transient
    - Behavior

Configurations

- Configuration
  - A set of concepts collectively providing required set of features
  - Feature set of configuration might be larger than sum of feature sets of all the concepts in the configuration
- Configurations of solution domain are identified during the solution domain analysis

Example solution domain model – features of configurations

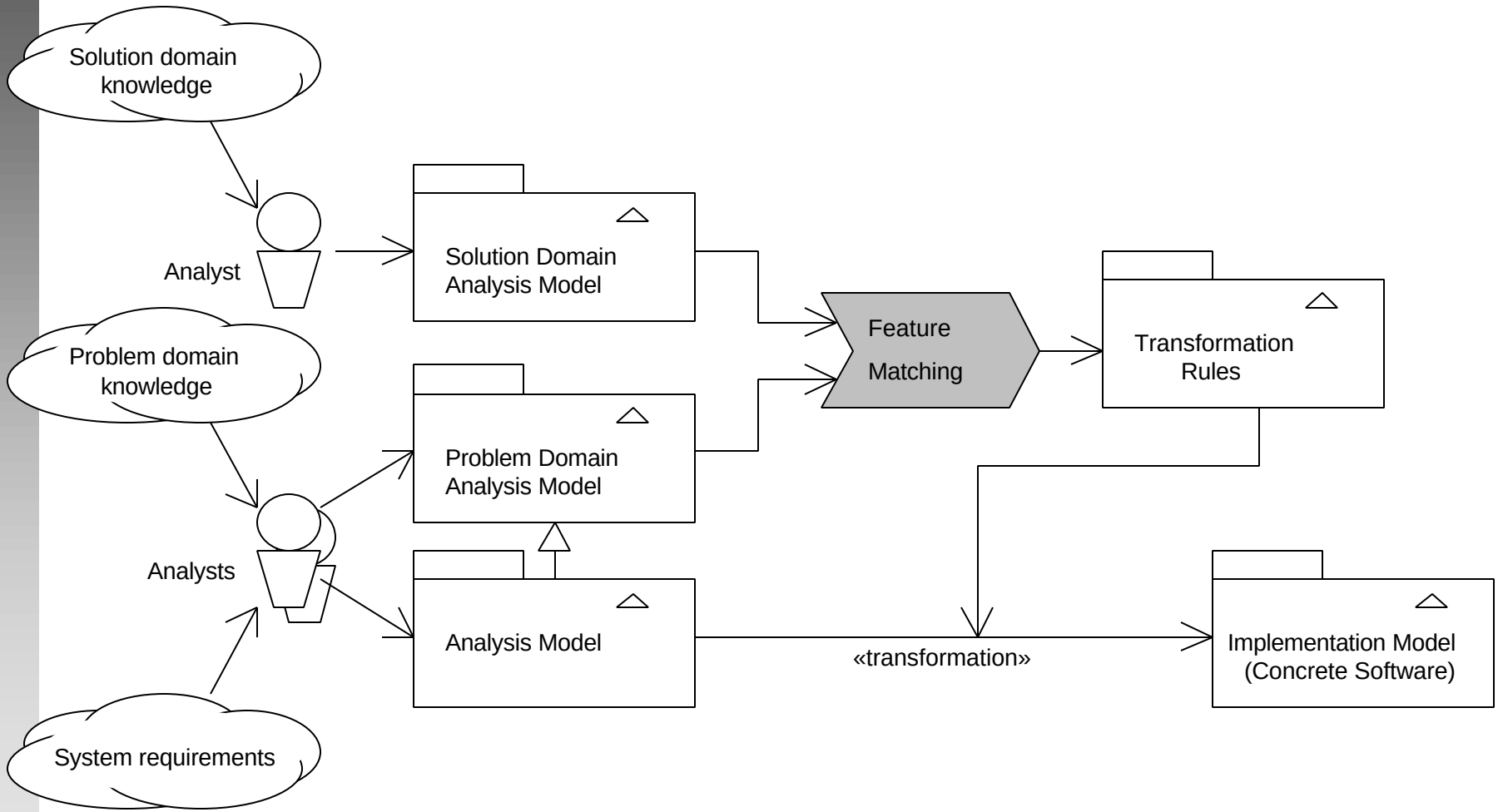
- Configuration
  {“JSP Page”, “Form Bean”, “Action”, “Entity Bean”}
  - Features (independent of domain)
    - Identity
    - State
    - Persistent → Storable, Searchable
    - Behavior
    - Viewable → Modifiable

- Configuration
  {“JSP Page”, “form Bean”, “Action”, “Session Bean”}
  - Features (independent of domain)
    - No identity
    - State is optional
    - Transient
    - Behavior

Contents

- Introduction
- Usage of models in software engineering
- Domain analysis
- Feature matching
  - Common feature space
  - Solution domain selection
  - Implementation synthesis
- Related work
- Conclusions

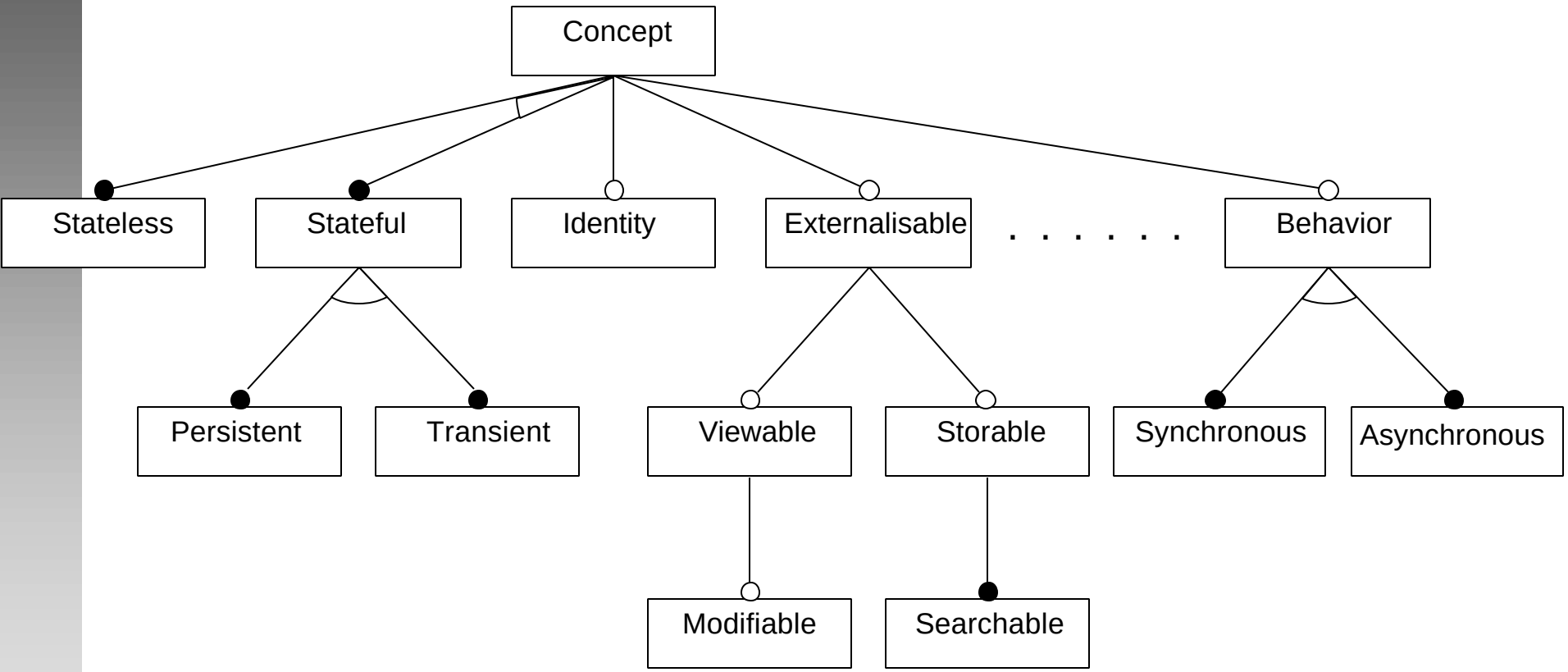
Feature matching in model-based software development



Common feature space

- Common features of concepts and configurations (identified for business information systems)
  - Functional features
    - May have identity
    - Independent | Dependent
    - Stateless | Stateful
    - Transient | Persistent → Storable, Searchable
    - Viewable → Modifiable
    - Business behavior → Asynchronous, Synchronous
    - …
  - Non-functional features
    - Efficiency → Speed, Space
    - Scalability
    - Modifiability
    - Portability
    - …

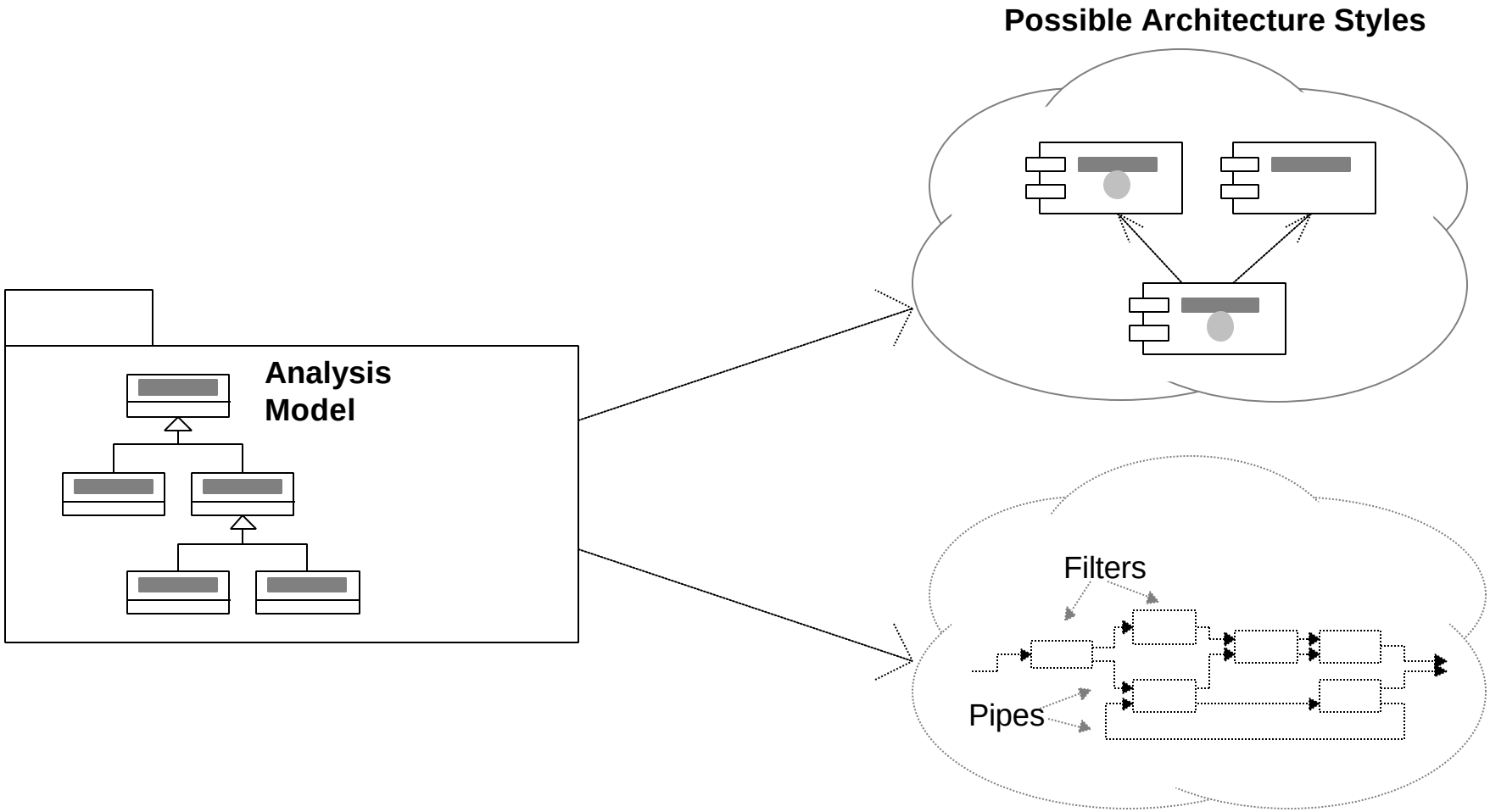
Feature diagram of common features of a concept



Solution domain and architecture selection

- Solution domain selection is based on the features offered by solution domain configurations
- Selecting the suitable architecture style
  - Based on functional features
    - Persistence
    - …
  - Based on non-functional features
    - Scalability
    - Modifiability
    - …
- Examples
  - Data-entry application → Central Repository
  - Signal processing application → Pipes and Filters
  - Decision Support → Blackboard

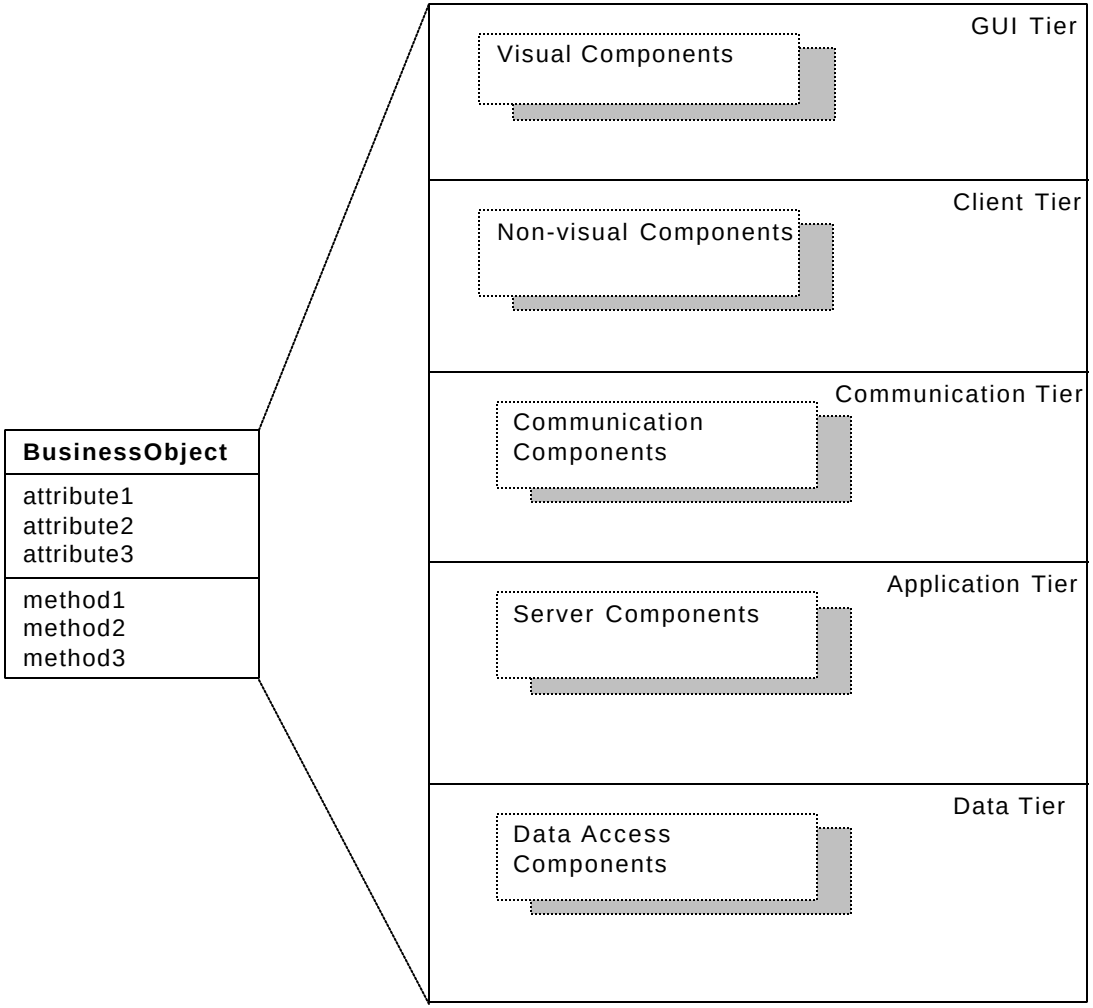
Mapping to different architecture styles



Implementation synthesis

- Selection of solution domain elements and configurations
  - For every problem domain element a suitable
    - solution domain element, or
    - configuration (set of solution domain elements)
  - is added to the implementation
- Suitability of solution domain element is decided by feature matching

Mapping of problem domain concept to solution domain configurations



Feature matching

- Concept descriptions
  - $C = F = \{f_i\}$
  - $\{C_1, \ldots, C_n\} = F \supseteq F_1 \cup \ldots \cup F_n$
- Mapping from problem to solution domain
  - $f : \{C^P\} \rightarrow \{C^S\}$
- Generic case
  - $F^P_1 \cup \ldots \cup F^P_n \subseteq F^S_1 \cup \ldots \cup F^S_m \Rightarrow \{C^P_1, \ldots, C^P_n\} \rightarrow \{C^S_1, \ldots, C^S_m\}$
- Trivial case
  - $F^P \subseteq F^S \Rightarrow \{C^P\} \rightarrow \{C^S\}$
- Complex cases
  - $F^P \subseteq F^S_1 \cup \ldots \cup F^S_m \Rightarrow \{C^P\} \rightarrow \{C^S_1, \ldots, C^S_m\}$
  - $F^P_1 \cup \ldots \cup F^P_n \subseteq F^S \Rightarrow \{C^P_1, \ldots, C^P_n\} \rightarrow \{C^S\}$

Strategies for feature matching

- Alternatives
  - $F^P \subseteq F^S_1$ & $F^P \subseteq F^S_2$
- Maximal additional features (greedy)
  - $F^S_1 \setminus F^P \subseteq F^S_2 \setminus F^P \Rightarrow \{C^P\} \rightarrow \{C^S_2\}$
- Minimal additional features
  - $F^S_1 \setminus F^P \subseteq F^S_2 \setminus F^P \Rightarrow \{C^P\} \rightarrow \{C^S_1\}$
- Optimal – cost function based
  - $\text{cost}(F^S_1 \setminus F^P) \leq \text{cost}(F^S_2 \setminus F^P) \Rightarrow \{C^P\} \rightarrow \{C^S_1\}$
- Cost function
  - Based on non-functional features of C^S_1

Related work

- Mapping to a predefined architecture
  - Mapping domain model to a generic design
    - (A. S. Peterson, J. L. Stanley, SEI, 1994)
    - Mapping domain analysis results (FODA or else) to predefined architecture (OCA – Object Connection Architecture) by architecture elements
  - FORM Feature-Oriented Reuse Method
    - (K. C. Kang, POSTECH, 1998)
    - Mapping feature space (FODA result) to predefined artifact space (architecture) by kinds of features
- Selection of architecture style
  - Attribute-Based Architecture Styles (ABAS)
    - (R. Kazman, L, Bass, et al., SEI, 1999)
    - Selection of architecture style based on reasoning about quality attributes

Conclusions

- Differences from other methods
  - Separate step of solution domain analysis
    - resulting reusable solution domain model
  - Common feature space for problem and domain analysis
  - Selection of solution domain and synthesis of implementation based on feature matching
- Next steps
  - Study of common feature space for problem and domain analysis (e.g. consistency, completeness)
  - Study of feature matching process
    - Creation of configurations with unanticipated features
  - Study of solution domain configurations (e.g. creation, sufficient set, relationship to design patterns)
  - Prototype implementation of feature matching algorithm

Thank You

Questions?

Practical Application

- Once&Done® software environment
  - OD Models
    - Extended meta-models
    - Reference models for insurance domains: Non-life (Property and Casualty), Life and Claims
  - OD Tools
    - Repository of models (extended meta-models)
    - Model combination tool
    - Rule driven generators
  - OD Framework
  - OD Process

Practical Application

- Once&Done® product-line (1995-2001)
  - 4 Systems for Property and Casualty Insurance
  - 3 Systems for Life Insurance
  - Claim Handling System
- Once&Done® models
  - Property and Casualty Reference Models
    - Private (380 entities + 394 relations)
    - Commercial (569 entities + 894 relations)
  - Life Reference Model
  - Claims Reference Model (96 entities + 43 relations)
- Change of technology
  - Client-Server → Three-tier → Web-based