



Design Patterns across the Modeling Process

PAME Workshop – MODELS 2016

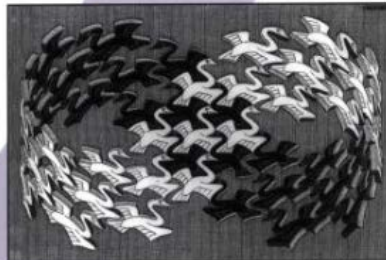
Jeff Gray
University of Alabama
Department of Computer Science

With special thanks to
Huseyin Ergin, Eugene Syriani, and Hyun Cho

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch



Copyrighted Material



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Average Customer Review: ★★★★★ (457 customer reviews)

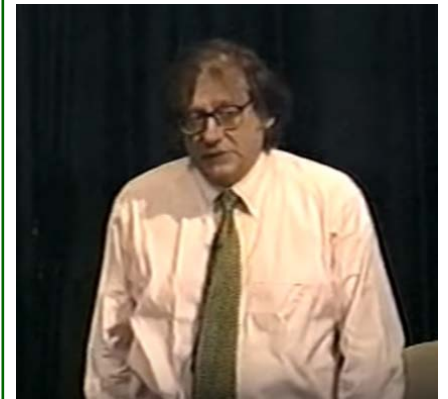
Amazon Best Sellers Rank: #5,758 in Books (See Top 100 in Books)

#1 in Books > Computers & Technology > Programming > Software Design, Testing & Engineering > **Software Reuse**
#1 in Books > Computers & Technology > Computer Science > AI & Machine Learning > **Computer Vision & Pattern Recognition**
#3 in Books > Textbooks > Computer Science > **Object-Oriented Software Design**

The “Alexandrian” Definition

*Each pattern describes a problem
which occurs over and over again in our
environment,
and then describes
the core of the solution to that problem,
in such a way that
you can use this solution a million times over,
without ever doing it the same way twice*

C. Alexander, “*The Timeless Way of Building*,” 1979



Must see video:

<http://bit.ly/alexander-oopsla96>

**PAME missed Alexander’s 80th
Birthday by one day!**

What Makes a Pattern a “Pattern” ?

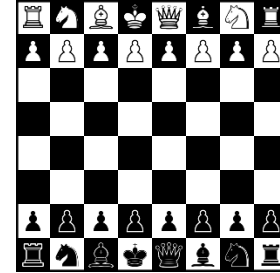
Pattern characteristics...







- ...it should solve a problem
 - i.e., it should be useful
- ...it has a definable context
 - it describes where the solution can be applied
- ...it is recurring
 - relevant in many situations
- ... instructional/educational
 - ▶ provides sufficient description to customize the solution
- ... distinct name
 - ▶ referred consistently

“Patterns don’t give you code you can drop into your application, they give you experience you can drop into your head.” *Patrick Logan*

An Analogy: Becoming a Chess Master

- *Learn the rules and physical requirements*
 - e.g., pieces, legal movements, chess board geometry and orientation.
- *Learn the principles*
 - e.g., relative value of certain pieces, strategic values of center squares, power of a threat etc.
- To become a master, one must *study the games of other masters*
 - these games contain patterns that must be understood, memorized and applied repeatedly
- There are hundreds, if not thousands, of such patterns

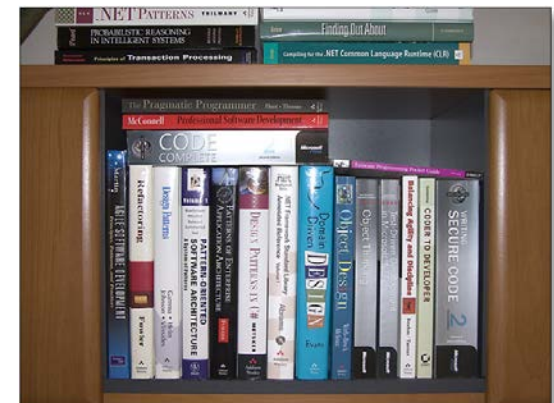
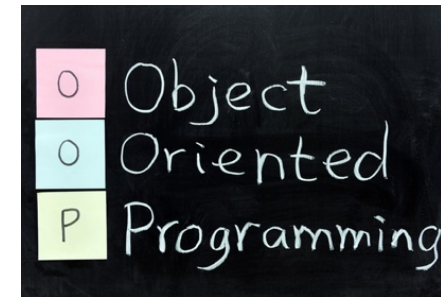


<i>Pieces and Point Value</i>		
<i>Pawn</i>		1
<i>Knight</i>		3
<i>Bishop</i>		3
<i>Rook</i>		5
<i>Queen</i>		9
<i>King</i>		<i>priceless</i>



Becoming a Software Design Master

- *First learn the rules*
 - e.g., the algorithms, data structures and languages of software
- *Then learn the principles*
 - e.g., principles that govern different programming paradigms
 - structured, modular, object-oriented, etc
- To truly master software design, one must *study the design of other masters*
 - these designs contain *patterns* that must be understood, memorized and applied repeatedly
- There are thousands of these patterns

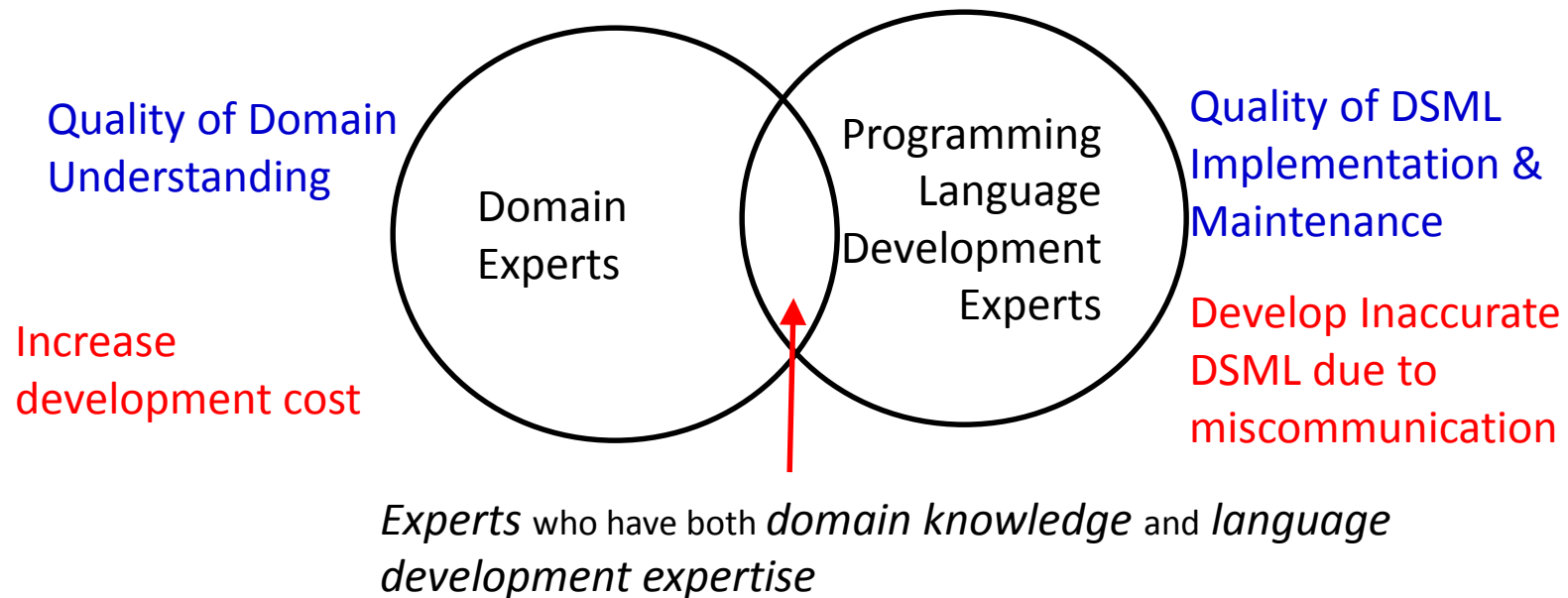


Patterns for Modeling

- What about modeling...
- Areas where students or first-learners may struggle, and could benefit from *experiential reuse*
 - Metamodel Patterns: assist students who are designing their first modeling language through a metamodel
 - Transformation Patterns: provide guidance to those learning to write model transformations

DSML Development Challenge

- Often requires familiarity of ***domain knowledge*** and ***language design expertise***



Metamodel Design Patterns

- *Metamodels can be designed by **reusing existing metamodel concepts** that represent commonly recurring metamodel design concepts across multiple domains.*
- *Such reuse of metamodeling experience may improve the quality of metamodel design as well as achieve a significant increase in productivity in the development of DSMLs. Perhaps most importantly, metamodel patterns provide a great educational/instructional tool for those new to MDE.*

An Approach for Identifying Metamodel Design Patterns

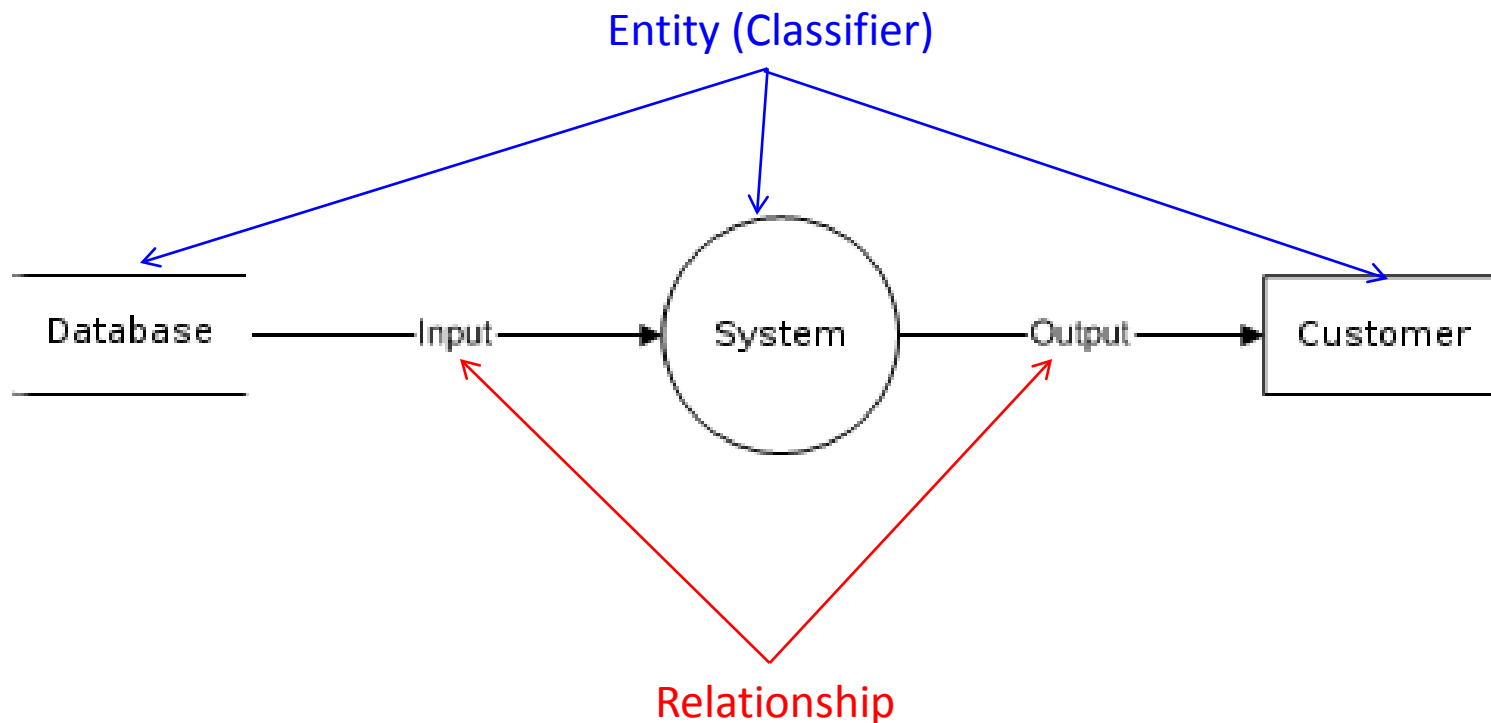
- Collect various types of DSMLs; examine concrete syntax of sample instances to observe emergent classifiers and relations
- Identify characteristics of each DSML and its modeling elements/relations
- Analyze commonality of DSMLs
- Identify candidate metamodel design problems
- Collect and review metamodel samples
- Propose metamodel design patterns

Collection of DSMLs Examined

Domain	Diagrams	Brief Description
Concurrent Discrete Event System Modeling	Petri Net	Modeling systems with concurrency and resource sharing
Data Modeling	ERD	Model the logical structure of database
Project Management	Gantt Chart	Model project activities with relevant information (i.e., duration, cost, ...)
	PERT Chart	Identify the critical path of the project by modeling the sequence of tasks
Electronic Circuit Design	Schematic Diagram	Represent how electronic components are connected with others
	PCB Layout	Show the placement of electronic components on printed circuit board
Molecular Modeling	-	Model the structures and reactions of molecules
SW Design	Flowchart	Model process or algorithm
	Component Diagram	Represent static structure of components and their relations
	UseCase Diagram	Describe system functionalities or behaviors with UseCase and Actor
	Class Diagram	Describe the static structure of the system in terms of classes

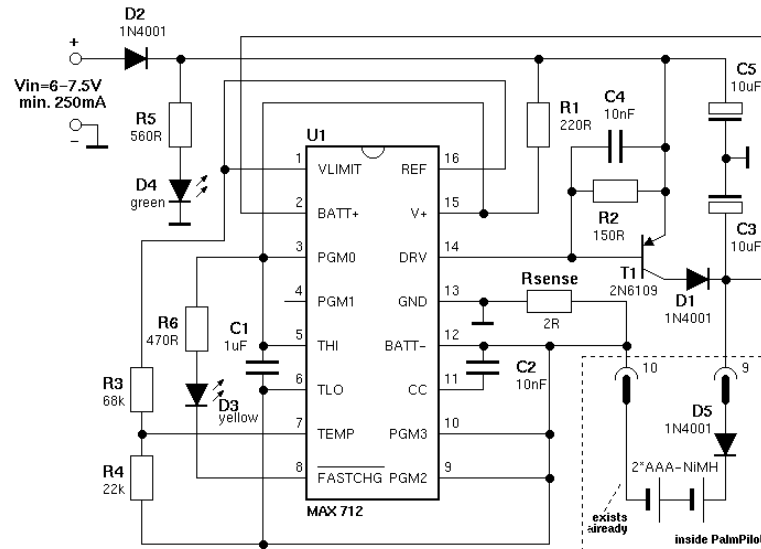
Identify Characteristics of DSMLs

- Classifiers and relations



Identify characteristics of DSMLs

- Electronic Circuit Design: Palm III Charger

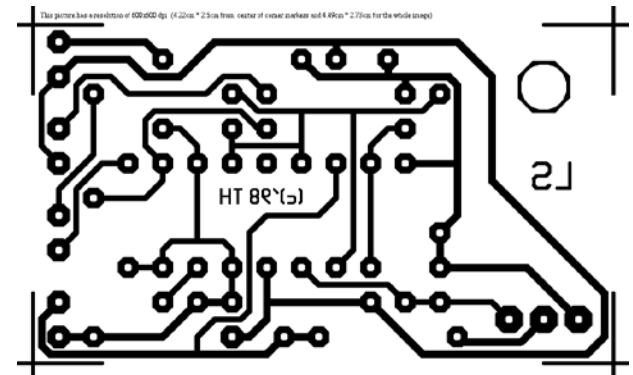


Schematic Diagram



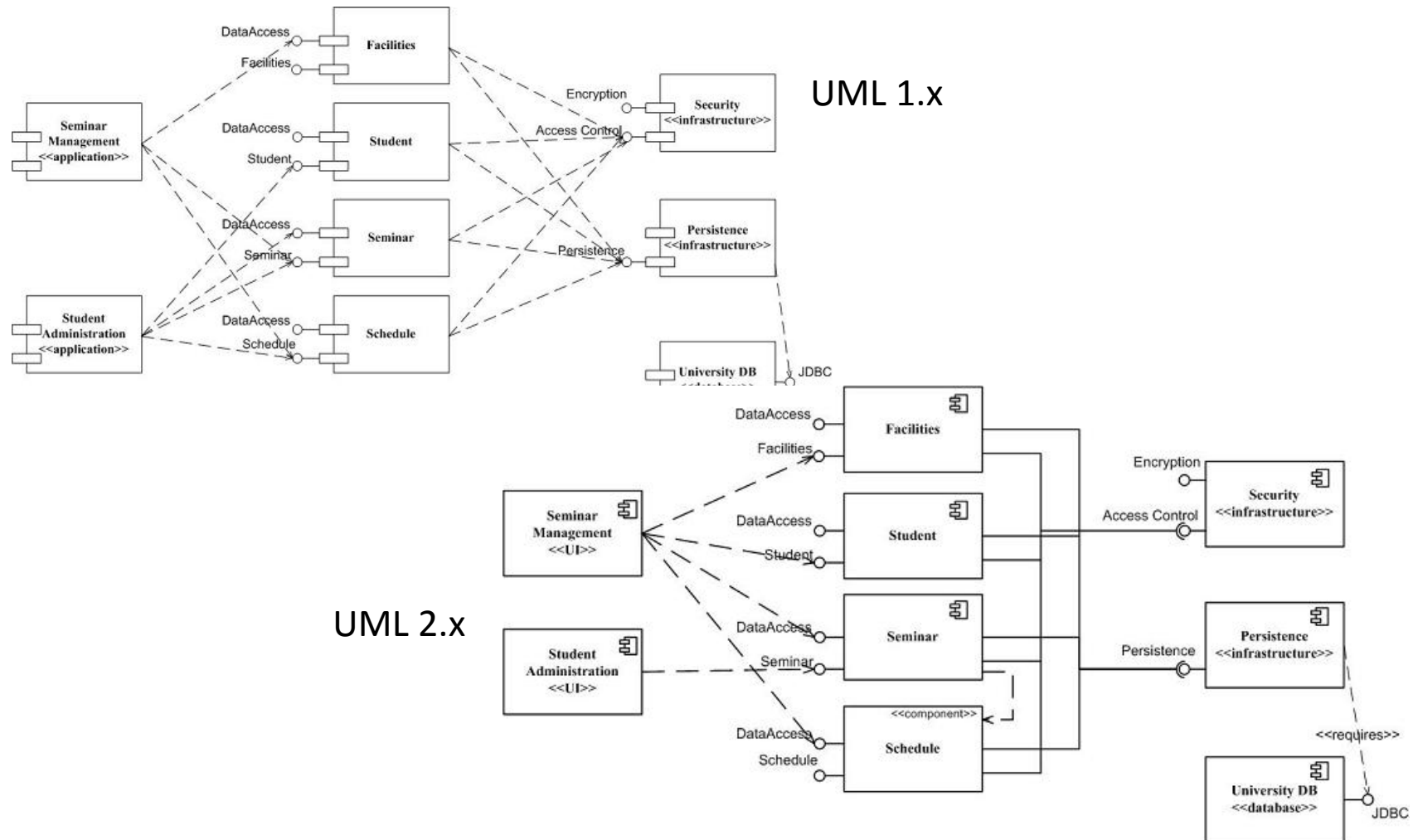
(c) 1999 Till Harbaum

PCB Layout Diagram



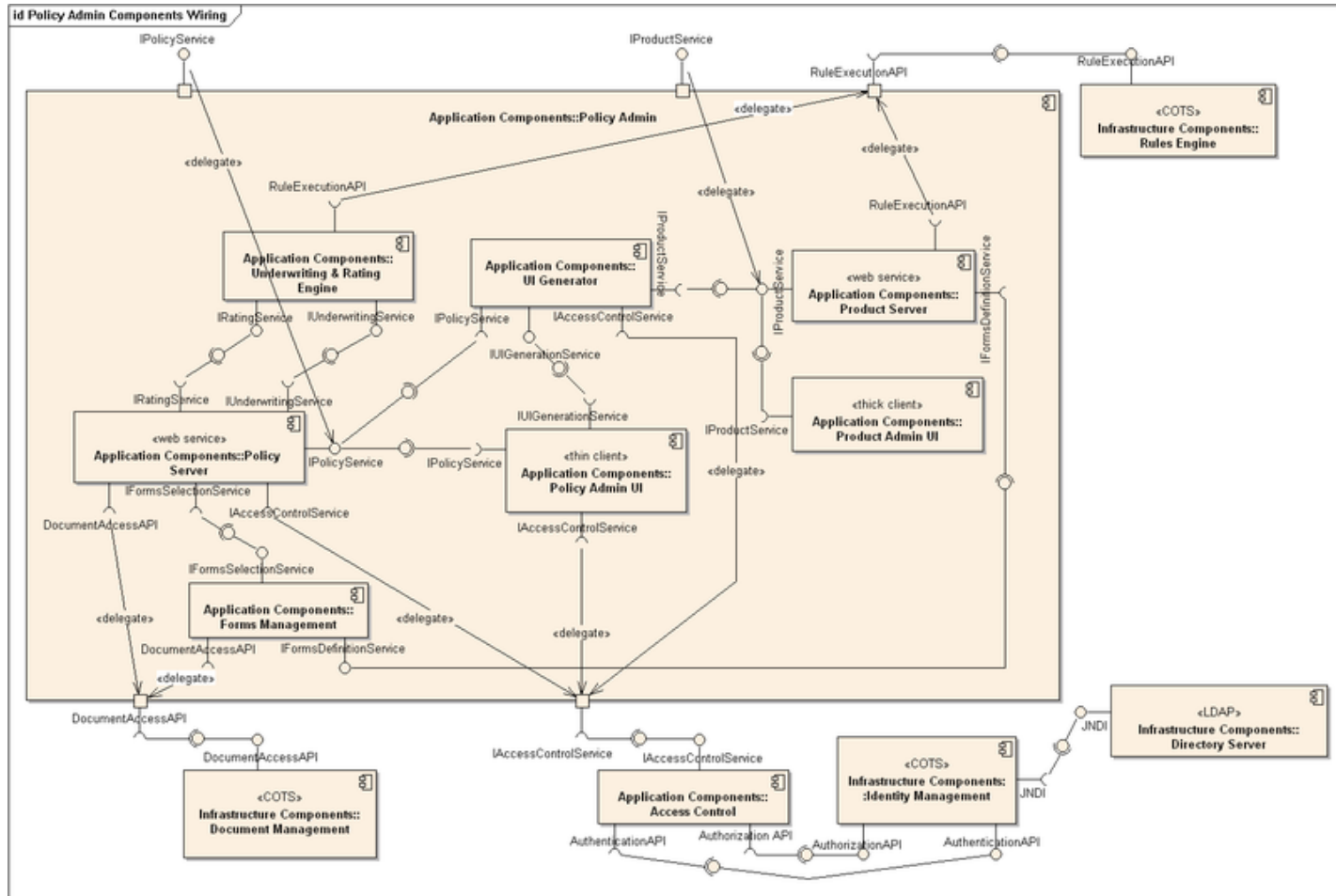
Identify characteristics of DSMLs (cont.)

- Component Diagram



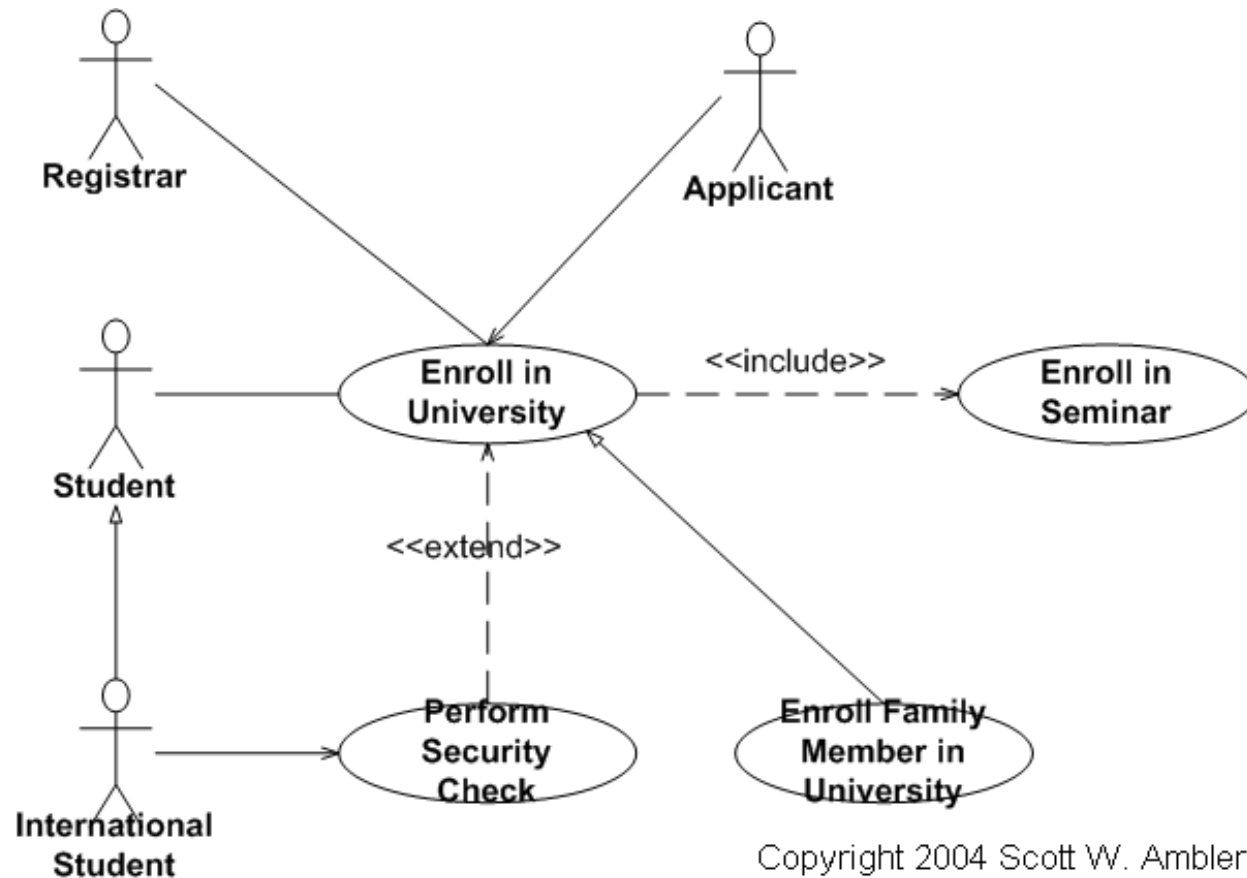
Identify characteristics of DSMLs (cont.)

- Component Diagram



Identify characteristics of DSMLs (cont.)

- Use Case Diagram



Features of DSMLs

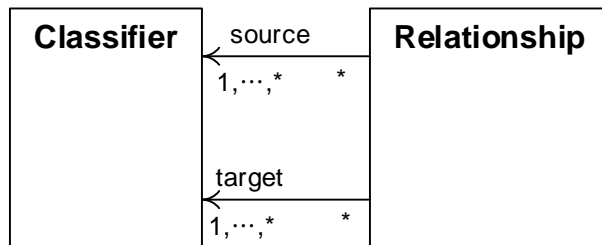


Questions for Identifying Candidate Patterns

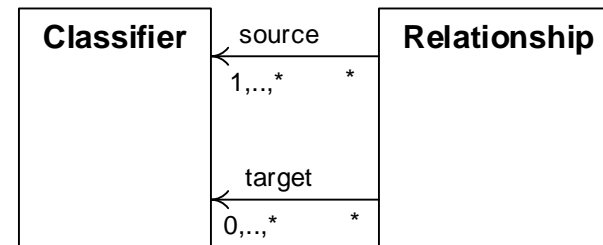
- What could be a primitive or base metamodel pattern, which could be common ground for metamodel design?
- How to extend the base metamodel if a DSML has complicated language constructs?
 - For example, a DSML can have typed relationships such as include and extend in a Use Case diagram.
- How to represent *boundedness* in the metamodel?
- How to design the metamodel to describe *containment* and *nesting*?

Base Metamodel Pattern

- What could be a primitive or base metamodel pattern, which could be common ground for metamodel design?
- How to represent *boundedness* in a metamodel?
- Applicable for simple Box-and-Line style DSMLs
 - Most common pattern for early stage of DSML development
 - Useful for Prototyping DSML



(a) Modified BPMN p92



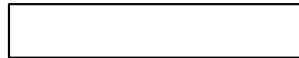
(b) Modified Bender et al.
metamodel design

Metamodel with (sub)types Pattern

- How to extend the base metamodel if a DSML has complicated language constructs?
- Extension of base metamodel design pattern
 - Add more expressiveness to DSMLs
 - Semantics of each relationship is required to enforce behaviors and properties
- Evaluation Point
 - Association point between Classifiers and Relationships

Containment/Nesting Pattern

- How to design the metamodel to describe *containment* and *nesting*?
- Some DSMLs may contain or nest modeling elements to control the abstraction level
 - Can focus on core intent by eliminating unnecessary details or give more descriptions by showing details



- Evaluation Point
 - Comprehensibility and Extendibility

Application of Metamodel Design Patterns

- Composition-based metamodel development

Expected benefits of Metamodel Design Patterns

- Avoid duplication of metamodel design for recurring design problems
- Keep high quality metamodel fragments
- Guide and Recognize key patterns and best-practices of metamodel design
- Reduce time-to-market for developing new DSMLs

Examples from Yesterday

- Antonio Garmendia's Doctoral Symposium talk
 - Patterns focused on scalability support for DSLs
 - Modularity
 - Filter
 - Scoping
 - Additional pattern of interest
 - Integrating different DSLs (Juergen Dingel)

Related Work – Metamodel Patterns

- Ana Pescador, Antonio Garmendia, Esther Guerra, Jesús Sánchez Cuadrado, and Juan de Lara, “Pattern-based Development of Domain-specific Modelling Languages,” *Model Driven Engineering Languages and Systems (MODELS)*, October 2015, Ottawa, Canada, pp. 166-175.
- Antonio Garmendia, Ana Pescador, Esther Guerra, Juan de Lara: Towards the Generation of Graphical Modelling Environments Aided by Patterns. SLATE 2015: 160-168

Model Transformation Design Patterns

- *Model transformations often have commonly recurring themes that can be captured as patterns. Idioms may also exist for specific transformation languages that can be described and reused.*
- *The reuse of transformation knowledge captured in design patterns may give insight to those who are learning a new transformation language and associated tooling.*

NOT ALWAYS EASY...

- Excerpts from solutions in various MTLs.

GrGen.NET^[1]

```
rule findCouples
{
  pn1:Person; pn2:Person;
  independent {
    pn1 -:personToMovie-> m1:Movie <=:personToMovie- pn2;
    pn1 -:personToMovie-> m2:Movie <=:personToMovie- pn2;
    pn1 -:personToMovie-> m3:Movie <=:personToMovie- pn2;
  }

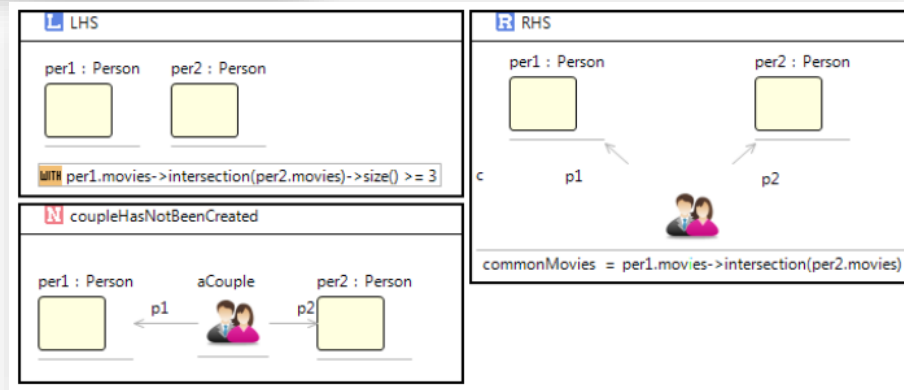
  modify {
    c: Couple;
    c -:p1-> pn1;
    c -:p2-> pn2;

    exec(addCommonMoviesAndComputeAverageRanking(c, pn1, pn2));
  }
} \ auto
```

FunnyQT^[2]

```
19 (defmacro define-group-rule [n]
20   (let [psyms (map #(symbol (str "p" %)) (range n))]
21     '(ip/defrule ~(symbol (str "make-groups-of-" n "!"))
22       {:forall true :no-result-vec true}
23       [~model ~'c]
24       [~'m<Movie>
25        :when (>= (person-count ~'m) ~n)
26        ~@(mapcat (fn [i]
27                    (let [ps (nth psyms i)]
28                      ['[~'m -<persons>-> ~ps
29                       :when (>= (movie-count ~ps) ~'c]
30                       ~@(when-not (zero? i)
31                           '[:when (neg? (compare (emf/egget-row ~(nth psyms (dec i)) :name)
32                                                                (emf/egget-row ~ps :name)))]))
33                        ~@(when-not (or (zero? i) (= 1 (dec n)))
34                            '[:when (n-common-movies? ~'c ~@(take (inc i) psyms))]]))
35                    (range n))
36          :when-let [~'cms (n-common-movies? ~'c ~@psyms)]
37          :as [~'cms ~@psyms]
38          :distinct]
```

e-Motions^[3]



[1] Geiß, R. and Kroll, M. (2008) GrGen. net: A fast, expressive, and general purpose graph rewrite tool. *Applications of Graph Transformations with Industrial Relevance*, pp. 568–569. Springer.

[2] Tassilo Horn. Model querying with funnyqt - (extended abstract). In Keith Duddy and Gerti Kappel, editors, ICMT, volume 7909 of Lecture Notes in Computer Science, pages 56–57. Springer, 2013.

[3] Rivera, J.E., Dur'an, F., Vallecillo, A.: On the behavioral semantics of real-time domain specific visual languages. In: WRLA. pp. 174–190 (2010)



TRANSFORMATION PATTERNS

Inexperienced model transformation developers may benefit from transformation design patterns

Requires:

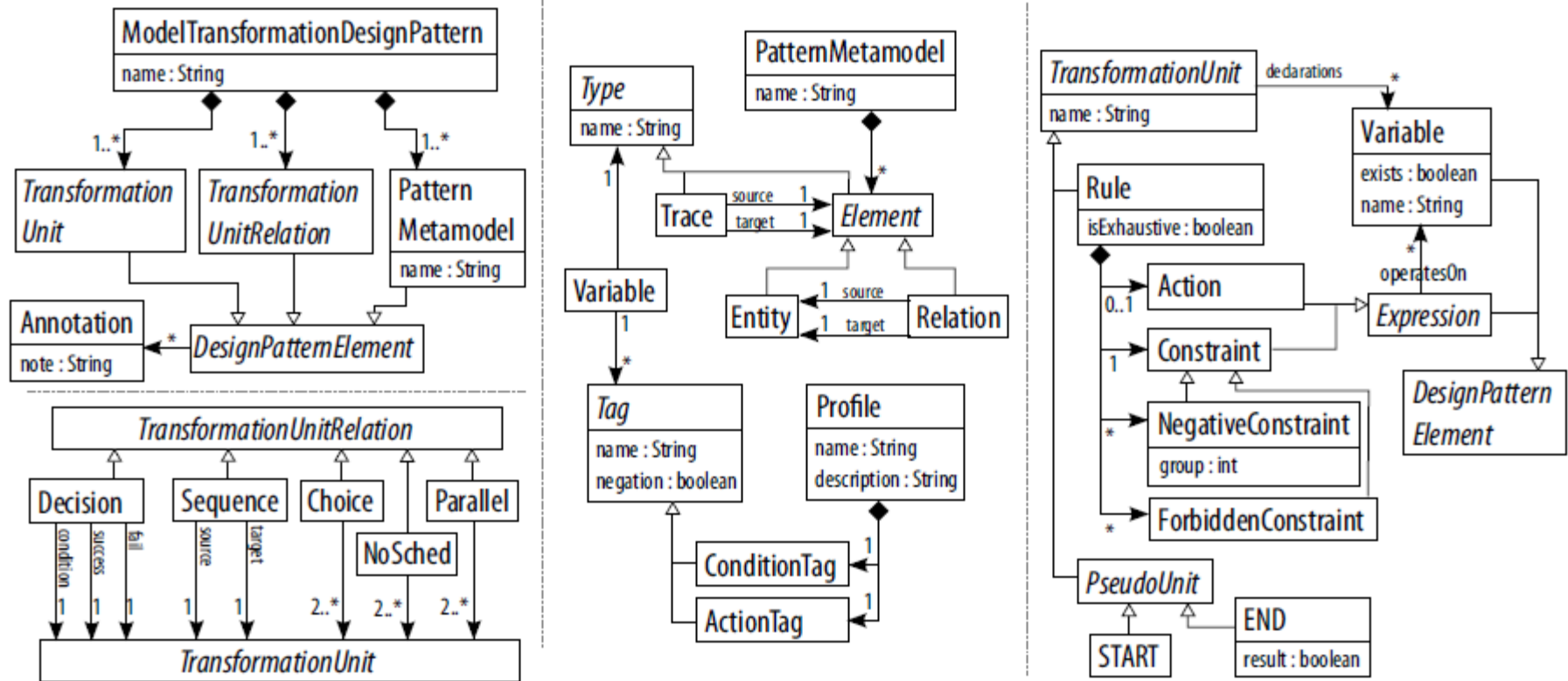
- 1. Finding the appropriate language to define model transformation design patterns.**
- 2. Identifying design patterns from existing transformation solutions.**



DELTA

- A language to express model transformation design patterns.
- Features:
 - Facilitate, reason, understand, document in a standard way^[1]
 - Independent from existing model transformation languages (MTL).
 - Helps to document and specify the entities and collaborations in a model transformation design pattern.
- DelTa (Design pattern language for model Transformation)
 - Offers concepts from existing MTLs.
 - Abstracts away MTL-specific concepts.
 - Expresses design patterns rather than model transformations.

DELTA METAMODEL



Huseyin Ergin, Eugene Syriani, and Jeff Gray, "Design Pattern Oriented Development of Model Transformations," *Computer Languages, Systems & Structures*, Volume 46, November 2016, pp. 106-139.

SAMPLE DESIGN PATTERN

- **Top-down Phased Construction^[1]:**

- **Summary:**

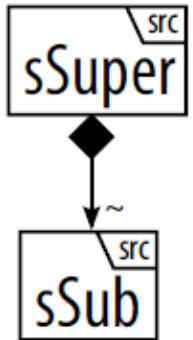
- Separates the transformation into phases depending on how the target model is composed.

- **Application conditions:**

- When there is a composition hierarchy in the source metamodel and the sub-element is mandatory in that relation.

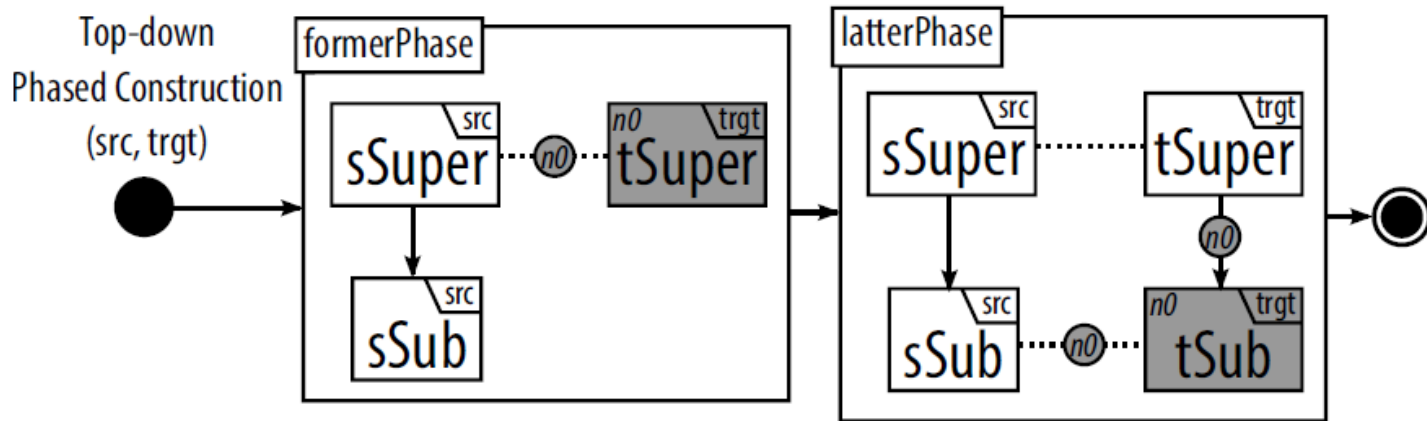
- **Examples:**

- The UML class diagram to relational database diagrams; we first use classes to create tables, and then use attributes to create columns.



SAMPLE DESIGN PATTERN

- **Solution:** The transformation is split into two phases. In the formerPhase rule, the target element corresponding to the super-element in the source is first created. In the latterPhase rule, target elements corresponding to the sub-elements in the source are then created.

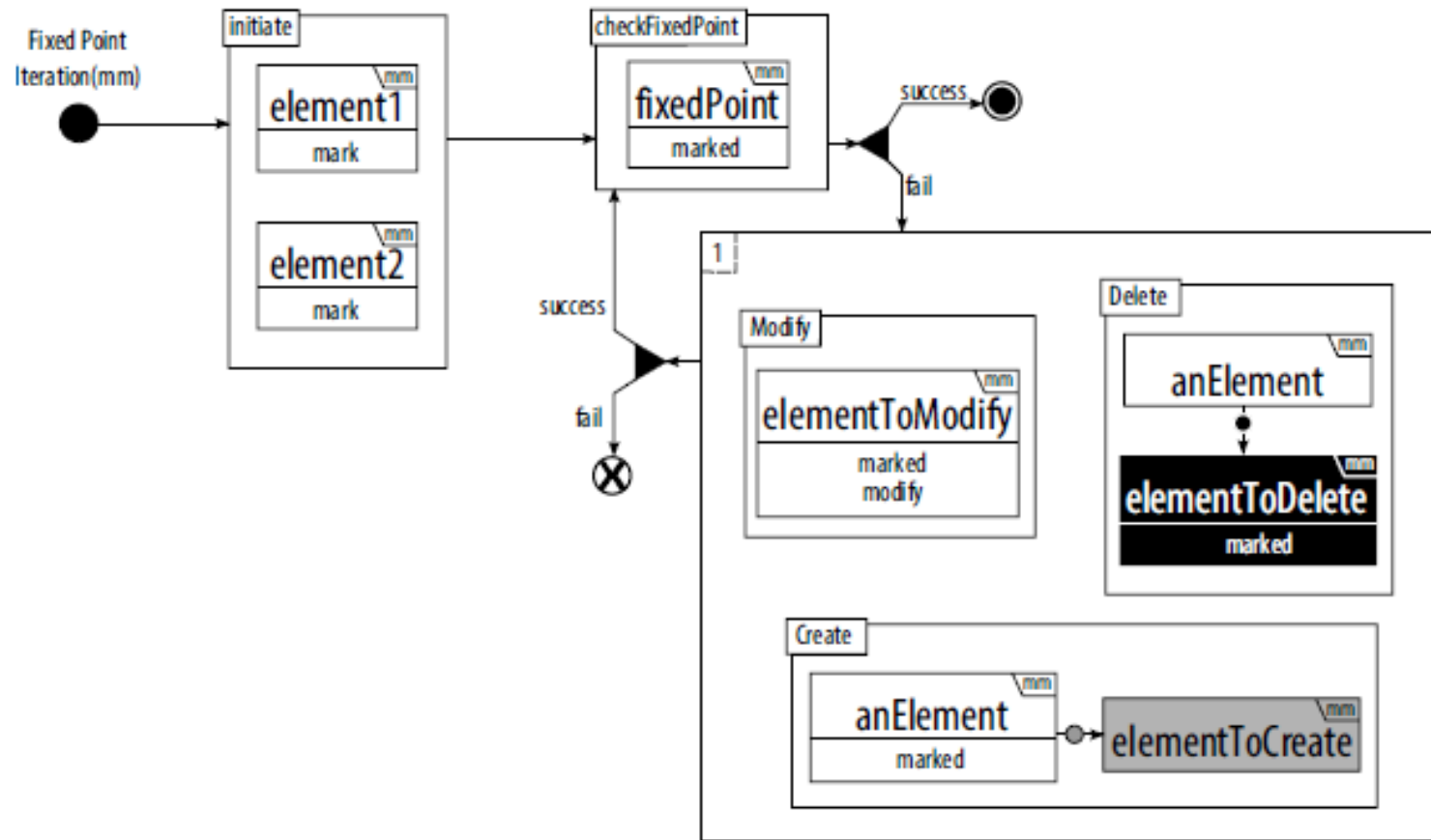


- **Benefits:** This pattern improves the modularity of the rules, letting each rule create one layer of target elements.
- **Disadvantages:** Because the rules are broken into phases, the number of overall rules will increase.

FIXED-POINT ITERATION PATTERN

- **Summary:**
 - Represents an iteration with a "do-until" loop structure.
 - Modifies the input model iteratively until a condition is satisfied.
- **Application Conditions:**
 - Applicable when the problem can be solved iteratively until a fixed point is reached.
 - Each iteration must perform the same modification on the model, possibly at different locations: either adding new elements, removing elements, or modifying attributes.
- **Examples:**
 - computing the lowest-common ancestor of nodes in a directed tree
 - finding the equivalent resistance in an electrical circuit
 - finding the shortest path using Dijkstra's algorithm

FIXED-POINT ITERATION PATTERN IN DELTA



FIXED-POINT ITERATION PATTERN

- **Summary:**
 - Represents an iteration with a "do-until" loop structure.
 - Modifies the input model iteratively until a condition is satisfied.
- **Application Conditions:**
 - Applicable when the problem can be solved iteratively until a fixed point is reached.
 - Each iteration must perform the same modification on the model, possibly at different locations: either adding new elements, removing elements, or modifying attributes.
- **Examples:**
 - computing the lowest-common ancestor of nodes in a directed tree
 - finding the equivalent resistance in an electrical circuit
 - finding the shortest path using Dijkstra's algorithm

TRANSFORMATION GENERATOR

DelTa Model Generator UI

Generate Models from Design Patterns

Selected design pattern: Entity Before Relation Show Design Pattern Details

Participant Customizations

Metamodels

src

trgt

Rule entityMapping

entityMapping

sEnt

tEnt

Rule relationMapping

relationMapping

sEnt2

tEnt2

relation sEnt sEnt2

relation tEnt tEnt2

Target language: GrGen.NET

Generate Transformation Model

Related Work – Transformation Patterns

- Agrawal, A. (2005) Reusable Idioms and Patterns in Graph Transformation Languages. *International Workshop on Graph-Based Tools, ENTCS*, **127**, pp. 181–192. Elsevier.
- Guerra, E., de Lara, J., Kolovos, D., Paige, R., and dos Santos, O. (2013) Engineering model transformations with transML. *Software and Systems Modeling*, **12**, 555–577.
- Lano, K., Kolaoudou-Rahimi, S., "Model-Transformation Design Patterns," *Software Engineering, IEEE Transactions on* , vol. 40, no. 12, December 2014, pp. 1224-1259.
 - A catalog of 29 transformation patterns
- Lano, K., Kolaoudou-Rahimi, S., Poernomo, I., Terrell, J., Zschaler, S., "Correct-by-construction Synthesis of Model Transformations using Transformation Patterns," *Software and System Modeling*, Volume 13, Number 2, June 2014, pp. 873-907.

Conclusion

- Defining metamodels for new languages, and the transformations that evolve their model instances, is not an easy task for inexperienced or new modelers
- The long-known benefits of design patterns seem to have an obvious advantage also with MDE
- Pedagogical needs for teaching MDE can be a source for additional investigation in MDE patterns
- Perhaps the main benefit will be the capture of experiential reuse, with automated tooling a secondary concern

Thank you for your attention



EXISTING TEMPLATES

Unified template	Bezivin	Levendovsky	Agrawal	Iacob	Lano	Ergin	GoF meaning
Summary	Motivation	Motivation	Motivation	Goal Motivation	Summary	Motivation	Intent
Application condition		Applicability	Applicability	Applicability	Application conditions	Applicability	Applicability
Solution	Solution	Structure	Structure	Specification	Solution	Structure	Structure
Benefits	Consequences	Consequences	Benefits		Benefits		Consequences
Disadvantages			Limitations		Disadvantages		
Example		Known uses	Known uses	Example	Application and examples	Examples	Known uses
Implementation						Implementation	Sample code
Related patterns		Variations			Related patterns	Variations	Implementation
							Related patterns

- Existing model transformation design pattern (MTDP) studies.
 - Not all of them call themselves MTDP though.
 - The separation between reusable idioms, patterns, design patterns.
- But the inconsistency is clear.
- Unification will help what is expected from a pattern and what are the outcomes.