A Formal Approach to Patterns in MDE

Yngve Lamo

Faculty of Engineering, Bergen University College, Norway

21 September 2010
VFFP 2010, Madrid, Spain
Outline

Introduction

Diagram Predicate Framework

Patterns

Model Transformations

Summary
Personal Background

- Master in mathematics NTNU
- Ph.D. in theoretical Computer Science UoB – Algebraic specifications of software, used mathematics such as: universal algebra, logic, category theory
- Get motivated to apply category theory in software engineering
  - In 2000 Uwe Wolter introduced me to the Generalized Sketches framework, created in Latvia in early 90’s, by Zinovy Diskin et al
  - In 2003 we started to use GS to formalize diagrammatic specification languages (such as UML, ER)
Why Category Theory?

- Category theory focuses on how objects are related to each other
  - black box view of systems
  - objects (systems) are described by their interface to other objects (systems)
  - quantification over objects (systems)
- Set theory focuses on how objects are created
  - white box view of systems
  - objects described by their internal structure
  - quantification over elements
Model-Driven Engineering (MDE)

- Software models as abstract representations of software systems
  - reasoning at a high level of abstraction
- Model-Driven Engineering (MDE)
  - models as primary artefacts of the software development process
  - generation of systems by means of model transformations
MDE Standards: State-of-the-art

- Modelling languages
  - Unified Modeling Language (UML)
  - Eclipse Modeling Framework (EMF)
  Usually graph-based languages
- Constraint languages
  - Object Constraint Language (OCL)
  Usually text-based languages
### Some Advantages of Diagrammatic Modelling

<table>
<thead>
<tr>
<th>Property</th>
<th>Advantage</th>
<th>Achieved by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation and communication</td>
<td>Facilitating intuitiveness</td>
<td>Visual models</td>
</tr>
<tr>
<td>Abstraction</td>
<td>Independence of the implementation platform</td>
<td>Abstract models and model transformations</td>
</tr>
<tr>
<td>Validation and verification</td>
<td>Revealing errors and flaws before the system is implemented</td>
<td>Formal models and model checking</td>
</tr>
</tbody>
</table>
Some facts about UML

- UML is de facto industry standard for modelling
- UML can only express constraints on binary relations, basically UML only specifies cardinality and typing constraints
- UML has serious issues regarding:
  - Semantics; UML models may be ambiguous and have semi-formal semantics
  - Complexity; UML uses 13 different types of diagrams
  - Expressibility; To express constraints over higher order relations one need to use string based logic, (Object Constraint Logic, OCL)
Formal modelling languages

- Nice semantics, several fundamental Software Engineering problems solved by use of formal methods
- Set based semantics (logic, algebras, etc.)
- No concept of meta modelling (index semantics)
- Hard for software engineers to apply in practice
- Lack of good software development tools based on formal approaches
- Only used by well trained experts
- High cost low productivity
Outline

Introduction

Diagram Predicate Framework

Patterns

Model Transformations

Summary
Project History

• 2003– Uwe Wolter and Yngve Lamo started to work with Generalized Sketches
• 2004-2006 Master student Ørjan Hatland implement prototype tool for GS (.NET framework)
• 2006-2010 PhD student Adrian Rutle, MDE, (meta)modeling, model transformation ⇒ DPF
• 2006-2008 Master student Stian Skjerveggen started to implement a graphical editor in Eclipse (GMF) framework
• 2008– PhD student Alessandro Rossini, Version Control for MDE
• 2009– Master Students Øyvind Bech, Dag Viggo Lokøen, tool support for DPF
• 2010– PhD student Florian Mantz, meta model evolution
Diagram Predicate Framework (DPF)

- A generalisation of categorical sketches
- Adaption of generalised sketches to MDE
- Graph based specification framework
  - models: graphs decorated with constraints
  - constraints formalised as categorical diagrams
DPF offers

- Diagrammatic formalisation of metamodelling
- Constraint-aware model transformation
  - constraints affect which structure to create in the target model
  - control which constraints to add to the created structure
- Today patterns in DPF:
  - pattern definition as diagrammatic specifications
  - pattern matching and application by model transformation
- Formalisation of version control of models
(Meta)modelling in OMG and DPF

Modelling approach

Sample model

• Model: abstract representation of a software system
(Meta)modelling in OMG and DPF

Modelling approach

Sample model

- Req 1: “a person must be child of exactly two persons”
(Meta)modelling in OMG and DPF

Modelling approach

- Model: specified by a modelling language

Sample model

OMG

Structural constraints

Person

childOf

OMG
(Meta)modelling in OMG and DPF

Modelling approach

- Modelling language: corresponding metamodel + semi-formal semantics

Sample model

UML

- Class
- Property
- Association
- Person

OMG
(Meta)modelling in OMG and DPF

Modelling approach

Sample model

- Req 2: “no person is his/her own child”
(Meta)modeling in OMG and DPF

Modelling approach

- Attached constraint: specified by means of text-based OCL
(Meta)modelling in OMG and DPF

Modelling approach

Sample model

- Diagram Predicate Framework (DPF)
(Meta)modelling in OMG and DPF

Modelling approach

- Modelling Language ↪ Metamodel ↪ Model
- Metamodel corresp. to OCL Constraints
- Model conforms to OCL Constraints
- OCL Constraints attached to Model
- Specification: graph + set of constraints

Sample model

- UML:
  - Class
  - Property
  - Association
- OCL:
  - Context Person
  - Inv: Irreflexive
  - Self.ChildOf->excluding(self)
  - Structural constraints
  - Attached OCL constraints
(Meta)modelling in OMG and DPF

Modelling approach

- Modelling Language -> Metamodel -> Model
- Metamodel corresp. to Metamodel
- Model conforms to OCL Constraints
- OCL Constraints attached to Model

Sample model

- UML
  - Class
  - Property
    - lower: Int
    - upper: Int
  - Association
  - Person
    - childOf
  - 2..2
  - Structural constraints
    - context Person inv: Irreflexive
    - self.ChildOf->excluding(self)

- OCL
- OMG
- DPF

- Specification: graph + set of constraints
(Meta)modelling in OMG and DPF

Modelling approach

- Modelling Language
- Metamodel
- Model
- OCL Constraints
- correspond to
- conforms to
- attached to

Modelling Language corresponds to Metamodel, which conforms to Model, which is attached to OCL Constraints.

Sample model

- UML
- Class
- Property
- Association
- lower: Int
- upper: Int
- Structural constraints
- Attached OCL constraints

OMG/DPF

- Specification: specified by a modelling formalism
(Meta)modelling in OMG and DPF

Modelling approach

- Modelling Language: Metamodel
  - Model
  - OCL Constraints
  - Specification
- Modelling Formalism: Meta-specification

Sample model

- UML
  - Class
  - Property
  - Association
  - Person
    - childOf
    - Structural constraints
    - Attached OCL constraints
      - context Person
      - inv: Irreflexive
      - self.ChildOf->excluding(self)
  - childOf
- OCL

- OMG
  - DPF

- Metamodel
  - Specification
  - OCL

- OCL
  - context Person
  - inv: Irreflexive
  - self.ChildOf->excluding(self)
(Meta)modelling in OMG and DPF

Modelling approach

- Modelling Language → Metamodel → Model
- Metamodel → OCL Constraints
- Modelling Language corresp. to Metamodelling
- OCL Constraints conforms to
- Metamodelling specified by
- Conforms to
- Attached to

Sample model

- UML Class → Property → Association
- Person 2..2
- Structural constraints: context Person
- OCL inv: Irreflexive
- Self.ChildOf→excluding(self)
- Attached OCL constraints

- Diagrammatic signature
- Class
- Person
- Reference
- Intended semantics
- \( p \)
- \([\text{mult}(n,m)]\)
- \( \forall x \in X : n \leq |f(x)| \leq m \)
- \([\text{irreflexive}]\)
- \( \forall x \in X : x \notin f(x) \)

- Modelling formalism: meta-specification + signature
Modelling approach

Sample model

- Integration of constraints
Sample Object-Oriented Modelling Hierarchy

- Modelling formalism: meta-specification + ...
Sample Object-Oriented Modelling Hierarchy

- Modelling formalism: meta-specification + signature
Sample Object-Oriented Modelling Hierarchy

- Req 1: “an employee must work for at least one department”
• Req 2: “a department may have none or many employees”
Sample Object-Oriented Modelling Hierarchy

- Req 3: “a project may involve none or many employees”
Sample Object-Oriented Modelling Hierarchy

- Req 4: “a project must be controlled by at least one department”
Req 5: “an employee involved in a project must work in the controlling department”
Sample Object-Oriented Modelling Hierarchy

$p \rightarrow \alpha^S_2(p)$

<table>
<thead>
<tr>
<th>p</th>
<th>$\alpha^S_2(p)$</th>
<th>Proposed vis.</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[mult(n,m)]</td>
<td>1 $\xrightarrow{f} 2$</td>
<td>$\chi \xrightarrow{[n:m]} \chi$</td>
<td>$\forall z \in X : n \leq</td>
</tr>
<tr>
<td>[surjective]</td>
<td>1 $\xrightarrow{f} 2$</td>
<td>$\chi \xrightarrow{[\text{sur}]} \chi$</td>
<td>$f(X) = Y$</td>
</tr>
<tr>
<td>[inverse]</td>
<td>1 $\xrightarrow{g} 2$</td>
<td>$\chi \xrightarrow{p(x)} \chi$</td>
<td>$\forall x \in X, \forall y \in Y : y \in f(x) \iff x \in g(y)$</td>
</tr>
<tr>
<td>[image-inclusion]</td>
<td>1 $\xrightarrow{g} 2$</td>
<td>$\chi \xrightarrow{V_x} \chi$</td>
<td>$\forall x \in X : f(x) \subseteq g(x)$</td>
</tr>
<tr>
<td>[composition]</td>
<td>1 $\xrightarrow{h} 2$</td>
<td>$\chi \xrightarrow{{g(y) \mid y \in f(x)}} \chi$</td>
<td>$\forall x \in X : f;g(x) = {g(y) \mid y \in f(x)}$</td>
</tr>
</tbody>
</table>
Sample Object-Oriented Modelling Hierarchy

- Invalid instance
Outline

Introduction

Diagram Predicate Framework

Patterns

Model Transformations

Summary
Patterns in Model Driven Engineering

- Patterns are usually explained in a semi formal or informal language.
- To be used in MDE patterns should be expressed formally facilitating:
  - model transformations and
  - automatic software development steps.
- To enhance usability one would like to have a visual language for pattern definition.
- Hence natural to start with a graph based approach.
- The Diagram Predicate Framework DPF is a graph based framework where one could specify constraints over arbitrary graphs.
Design Patterns

• A solution strategy for a common problem (e.g. facade, decorator, singleton, etc.)
  • Reusable
  • Abstract
  • Conceptually clear
    • the pattern should be understand by referring to it’s name

• Often describes a solution for a part of a bigger system.

• Usage scenarios:
  1. guideline for design (specification)
  2. check whether a design conforms to a pattern
  3. if not, change the design (pattern enforcement by transformation/refactoring)
• Facade Design Pattern: Interface for subsystems
One aspect is to design a specifications that satisfies the pattern
• pattern applied to check existing specifications ...
... pattern enforcement by defining a transformation (refactoring) rule which has the design pattern as its RHS
• Match finding: is the pre-condition (LHS) of the rule satisfied?
• Apply pattern: by applying the transformation rule
Pattern as Model Transformation

- Specification of pattern
- Define refactoring rule
- Model refactoring is done “conceptually” in two steps:
  - Step one adding structure by using pushout construction
  - Step two deleting structure by pullback construction
Outline

Introduction

Diagram Predicate Framework

Patterns

Model Transformations

Summary
Model Transformation Approach

- Given source and target modelling formalisms and a source model ...
• ... we want to generate a target model
Model Transformation Approach

- How we get the target model from the source model?
Model Transformation Approach

- We have to relate the modelling formalisms
Model Transformation Approach

- First: Define morphisms to an appropriate joined modelling formalism
Model Transformation Approach

- Second: Define constraint-aware transformation rules
Model Transformation Approach

- Third: Apply the model transformation
• Third: Apply the model transformation  
  • A: Convert the source model to an intermediate model
Model Transformation Approach

- Third: Apply the model transformation
  - B: Apply the transformation rules (Pushout)
Model Transformation Approach

- Third: Apply the model transformation
  - C: Project out the target model (Pullback)
Model Transformation Approach

- Heterogeneous, out-place model transformation
Step 1: Joining Modelling Formalisms
Step 1: Joining Modelling Formalisms
Step 2: Define Constraint-Aware Rules

- Rule $r_1$. Class to table and primary key

<table>
<thead>
<tr>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:Class</td>
<td>1:Class $\rightarrow$ 1:Table $\rightarrow$ Int:DT$^t$</td>
</tr>
</tbody>
</table>

Diagram:

- $L$: 1:Class
- $R$: 1:Class $\rightarrow$ 1:Table $\rightarrow$ Int:DT$^t$
Step 2: Define Constraint-Aware Rules

- Rule \( r_2 \). Attribute to column
Step 2: Define Constraint-Aware Rules

- Rule $r_3$. Many-to-one references to foreign key
Step 2: Define Constraint-Aware Rules

- Rule $r_4$. Many-to-many references to link table and foreign key
Step 2: Define Constraint-Aware Rules

- Rule $r_5$. Many-to-many reference to link table and foreign keys
Step 2: Define Constraint-Aware Rules

- Rule \( r_6 \): [inverse] and [surjective] to [foreign-key], [image-equal] and [jointly-injective]
Step 3: Applying Model Transformation
A: Type Conversion of Source Model

Diagram:

- Class
- Data Type
- Employee
- Department
- Project

Relationships:
- empDeps \([1..\infty]\) from Employee to Department
- [inv] from Department to Employee
- depEmps \([1..\infty]\) from Department to Employee
- proDeps:depEmps from Project to Employee
- proEmps from Employee to Project
- proDeps from Department to Project

Attributes:
- Reference
- Attribute
A: Type Conversion of Source Model
B: Applying the Rules (after $r_5$)

Blue: Matched, Green: Added
**B: Applying the Rules (after \( r_6 \))**

Blue: Matched, Green: Added
Last Intermediate Model
Outline

Introduction

Diagram Predicate Framework

Patterns

Model Transformations

Summary
Diagram Predicate Framework (DPF)
- A “pattern” for definition of diagrammatic specification formalisms
- Integration of constraints in (meta)modelling

Pattern as model transformation

Constraint-aware model transformation
- Source constraints transformed to target constraints
- Constraints used to control
  - which structure to create in the target model
  - which constraints to add to the created structure
Summary

- Diagram Predicate Framework (DPF)
  - A “pattern” for definition of diagrammatic specification formalisms
  - Integration of constraints in (meta)modelling
- Pattern as model transformation
- Constraint-aware model transformation
  - Source constraints transformed to target constraints
  - Constraints used to control
    - which structure to create in the target model
    - which constraints to add to the created structure
- Out-place approach to heterogeneous model transformation
Related work

- Graph Transformation System (GTS) *Hartmut Ehrig et al*
  - Adding support for transforming constraints
- Triple Graph Grammar (TGG) *Schürr et al*
  - Adding support for diagrammatic constraints in the joined modelling formalism
- Formalisation of Patterns *De Lara et al*
  - Pattern-based model-to-model transformations
  - GoF design patterns
  - Using Triple Graph Grammars
Further reading

Thank you!

Questions?
Challenges in modelling

- Mixing graph-based structures with textual constraints
  - Different technical spaces
    - checking models in two different engines/steps
    - model-constraint synchronisation problem
    - violation of “everything-is-a-model” vision of MDE
  - Challenge for domain experts who do not understand OCL
Formalisation approach

• Based on category theory
  • Sketches formalism: define semantics of diagrams (thus graph-based models)
    • models: graphs (nodes and edges)
    • model properties: universal properties (limit, colimit, commutative diagrams)
  • Generalized sketches formalism
    • not only universal properties
    • user-defined diagrammatic predicate signatures
• DPF: specification formalism based on generalized sketches
## Sample Signature for Relational Data Models

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\alpha^{\oplus 2}(p)$</th>
<th>Proposed vis.</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[primary-key]</td>
<td>1 $\xrightarrow{f}^{}$ 2</td>
<td>$X \xrightarrow{f}^{} [pk] Y$</td>
<td>$f$ is [total] and [injective]</td>
</tr>
<tr>
<td>[foreign-key]</td>
<td>1 $\xrightarrow{f}^{}$ 2 $\xrightarrow{g}^{}$ 3</td>
<td>$X \xrightarrow{f}^{} Y \xrightarrow{g}^{} Z$</td>
<td>$f(X) \subseteq g(Y)$</td>
</tr>
<tr>
<td>[image-equal]</td>
<td>1 $\xrightarrow{f}^{}$ 2 $\xrightarrow{g}^{}$ 3</td>
<td>$X \xrightarrow{f}^{} Y \xrightarrow{g}^{} Z$</td>
<td>$f(X) = g(Z)$</td>
</tr>
<tr>
<td>[join]</td>
<td>1 $\xrightarrow{f}^{}$ 2 $\xrightarrow{g}^{}$ 3 $\xrightarrow{f'}^{}$ 4</td>
<td>$X \xrightarrow{f}^{} Y \xrightarrow{g}^{} Z \xrightarrow{f'}^{} XZ$</td>
<td>$\forall x \in X, \forall z \in Z : (x, z) \in XZ$ iff $f(x) = g(z)$</td>
</tr>
</tbody>
</table>