Towards a Sketch based Specification Framework for Software Engineering

Yngve Lamo

Department of Computer Engineering
Faculty of Engineering
Bergen University College
NORWAY

11.05.2009
Algebra Seminar
Department of Mathematical Sciences
NTNU, Trondheim, NORWAY
Outline

1 Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2 Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3 Applications
   - Metamodeling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
Outline

1. Introduction and Motivation
   - Background
     - Algebraic Specifications
     - Object Orientation

2. Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
     - Diagrammatic Specification
     - Instance

3. Applications
   - Metamodeling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
- Master of mathematics NTNU - 1996
  - Geometrical function theory
- Started to study Computer Science in 1997
  - Ph.D. in theoretical Computer Science 2003
  - Thesis in the field of Algebraic specifications of software:
    Institution of Multialgebras, used mathematics such as:
    universal algebra, logic, category theory
- Get motivated to apply category theory to software engineering
  - In 2000 Uwe Wolter introduced me to the Generalized Sketches framework, created in Latvia in early 90’s, by Diskin ++
  - In 2003 we started to work on formalizing diagrammatic specification languages (such as UML)
Outline

1. Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2. Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3. Applications
   - Metamodelling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
Algebraic Specifications

- Algebraic specifications \((\Sigma(S, \Omega), \Phi)\)
- Signature \(\Sigma\) consist of a set of sort symbols \(S\), and \(S\) sorted operation symbols \(\Omega\). Constraints are specified by logical axioms \(\Phi\)
- Variation in semantics, \(S\) interpreted as Sets, operations \(S\)-sorted (partial/multi valued) functions, \(\Phi\) a subset of first order logic with equality.
- Semantic often restricted to classes of algebras with initial object.
  - Initial Term Algebra \(\sim\) Computation
- Algebras is well suited to specify abstract data types ADTs
### List of Elements

<table>
<thead>
<tr>
<th>$S$</th>
<th>{List, El}</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega$</td>
<td>empty: $\rightarrow$ List</td>
</tr>
<tr>
<td></td>
<td>head: List $\rightarrow$ Element</td>
</tr>
<tr>
<td></td>
<td>tail: List $\rightarrow$ List</td>
</tr>
<tr>
<td></td>
<td>append: El $\times$ List $\rightarrow$ List</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>head(append(x,l)) = x</td>
</tr>
<tr>
<td></td>
<td>tail(append(x,l)) = l</td>
</tr>
</tbody>
</table>
Algebraic Specifications observations

- Nearly as many formalisms as researchers in the field
- Each formalism tends to have the same expressive power
- The interesting thing from a CS view is how to express properties and not only what could be expressed
- In programming theory on tries to find the right abstraction
- Algebras are not suited for object oriented modeling
- Algebras are difficult for programmers to understand
Outline

1. Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2. Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3. Applications
   - Metamodeling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
Object orientation and Object Oriented modeling

- Object Orientation, Norwegian contribution to computer science
- Object Orientation (OO) is about: Classes, Objects, References, Attributes, Instances (Note that we will not talk about dynamical aspect such as state and processes)
- OO modeling, graphical representation of software systems (UML, ER, etc...)
- OO models has semiformal semantics
- Conceptual mismatch with Algebras (Set and Functions)
- We propose a formal framework for OO modeling based on sketches (graphs)
Project goal

- To apply and adapt sketch based methods to software engineering
  - Develop a framework to express software models and model transformations
  - Develop tools that justifies the framework
Outline

1 Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2 Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3 Applications
   - Metamodeling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
DPF

- Aims to combine the intuition from graphical modeling languages with the semantic rigor of formal methods
- Based on Generalized Sketches
- Sketches is a graphical representation of category theory, constraints expressed by universal constructions (limits, co-limits and commutative) diagrams
- Generalized Sketches is sketches extended with constraints on arbitrary diagrams, not only universal constructions
- In DPF, sketches are generalized and adapted to software engineering
Outline

1. Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2. Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3. Applications
   - Metamodeling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
Diagrammatic specifications

- **Signature** $\Sigma(\Pi, \alpha)$
  - Collection of constraint (predicate) names $\Pi$ with
  - An arity function $\alpha : \Pi \rightarrow \text{graph}$

- **Diagrammatic Specifications** consist of
  - the underlying graph $G(S)$ together with a
  - set of constraints $C_S$ i.e. diagrams $\delta : \alpha(c_S) \rightarrow G$, where $c_S \in \Pi$
  - Formulas over constraints
<table>
<thead>
<tr>
<th>(\Pi)</th>
<th>(\alpha)</th>
<th>visualization</th>
<th>intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[total]</td>
<td>1 (\xrightarrow{x} 2)</td>
<td><img src="signature_example.png" alt="Signature - Example" /></td>
<td>(\forall a \in A :</td>
</tr>
<tr>
<td>[key]</td>
<td>1 (\xrightarrow{x} 2)</td>
<td><img src="signature_example.png" alt="Signature - Example" /></td>
<td>(\forall a, a' \in A : a \neq a' \implies f(a) \neq f(a'))</td>
</tr>
<tr>
<td>[inv]</td>
<td>1 (\xrightarrow{x} 2)</td>
<td><img src="signature_example.png" alt="Signature - Example" /></td>
<td>(\forall a \in A, \forall b \in B : b \in f(a) \iff a \in g(b))</td>
</tr>
<tr>
<td>[jointly-key] or [1-1]</td>
<td>1 (\xrightarrow{x} 2)</td>
<td><img src="signature_example.png" alt="Signature - Example" /></td>
<td>(\forall a, a' \in A : a \neq a' \implies f(a) \neq f(a') ) or (g(a) \neq g(a'))</td>
</tr>
</tbody>
</table>
Diagram Predicate Framework (DPF)

Diagrammatic Specification – Example

```
Person
  ^
  |[key]
  | p_name
(1)

Company
  ^
  |[c_name]
  | address
(1)

Employment
  ^
  |[1-1]
  | employee
(1)

  ^
  | employer
(1)

  ^
  | [start_date]
  | 1

  ^
  | [salary]
  | 1

[Date]
[Int]
```
Outline

1 Introduction and Motivation
   • Background
   • Algebraic Specifications
   • Object Orientation

2 Diagram Predicate Framework
   • Diagram Predicate Framework (DPF)
   • Diagrammatic Specification
   • Instance

3 Applications
   • Metamodeling
   • Tools
   • Model-driven Engineering (MDE)
   • Model Transformations
Instance of Diagrammatic Specification – Example

Diagram Predicate Framework (DPF)

Instance

Diagram Predicate Framework (DPF)

Applications

Diagram Predicate Framework (DPF)

Diagrammatic Specification

Instance

Yngve Lamo

Diagram Predicate Framework (DPF)
The semantic interpretation of $\Sigma := (\Pi, \alpha)$ assigns to each $p \in \Pi$ a set $[p]$ of graph homomorphisms $\tau : O \rightarrow \alpha(p)$, where $O$ may vary over all graphs.

An instance of a specification $S$ is a graph $I$ together with a graph homomorphism $\iota : I \rightarrow G(S)$, such that for each constraint $(p, \delta) \in S(\Pi)$, $\iota^* \in [p]$, where $\iota^* : O^* \rightarrow \alpha(p)$ is given by the pullback diagram:

$$
\begin{array}{ccc}
\alpha(p) & \xrightarrow{\delta} & G(S) \\
\uparrow \iota^* & \downarrow \ & \uparrow \iota \\
O^* & \xrightarrow{\delta^*} & I \\
\end{array}
$$
Outline

1. Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2. Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3. Applications
   - Metamodeling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
### OMG Metamodel Levels

<table>
<thead>
<tr>
<th>OMG levels</th>
<th>OMG Standards/examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_3$: Meta-metamodel</td>
<td>MOF</td>
</tr>
<tr>
<td>$M_2$: Metamodel</td>
<td>UML language</td>
</tr>
<tr>
<td>$M_1$: Model</td>
<td>A UML model: Class &quot;Person&quot; with attributes &quot;name&quot; and &quot;address&quot;</td>
</tr>
<tr>
<td>$M_0$: Instance</td>
<td>An instance of &quot;Person&quot;: &quot;Ola Nordmann&quot; living in &quot;Sotraveien 1, Bergen&quot;</td>
</tr>
</tbody>
</table>
## Diagram Predicate Framework

<table>
<thead>
<tr>
<th>OMG’s own layers</th>
<th>OMG’s layer in DPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_3 )</td>
<td>( S_3(\Pi_3) )</td>
</tr>
<tr>
<td>( M_2 )</td>
<td>( S_2(\Pi_2) )</td>
</tr>
<tr>
<td>( M_1 )</td>
<td>( S_1(\Pi_1) )</td>
</tr>
<tr>
<td>( M_0 )</td>
<td>( S_0 )</td>
</tr>
</tbody>
</table>

\( \Pi_3 \rightarrow G(S_3) \)
\( \Pi_2 \rightarrow G(S_2) \)
\( \Pi_1 \rightarrow G(S_1) \)

Yngve Lamo

Diagram Predicate Framework (DPF)
Definition (Conformance)

Given signatures \( \Sigma_1 = (\Pi_1, \alpha_1) \), \( \Sigma_2 = (\Pi_2, \alpha_2) \) and a \( \Sigma_2 \)-specification \( S_2 = (G(S_2), S_2(\Pi_2)) \), a \( \Sigma_1 \)-specification \( S_1 = (G(S_1), S_1(\Pi_1)) \) *conforms to* \( S_2 \) if there exists a graph homomorphism \( \iota_{S_1} : G(S_1) \to G(S_2) \) such that \( (G(S_1), \iota_{S_1}) \) is an instance of \( S_2 \).

\[
\begin{align*}
\Pi_2 & \quad \xrightarrow{S_2(\Pi_2)} \quad G(S_2) \\
\Pi_1 & \quad \xrightarrow{S_1(\Pi_1)} \quad G(S_1)
\end{align*}
\]

\( \iota_{S_1} \)
Definition (Modelling Formalism)

A modelling formalism \( F = (\Sigma_1, S_2, \Sigma_2) \) is given by signatures \( \Sigma_1 = (\Pi_1, \alpha_1) \) and \( \Sigma_2 = (\Pi_2, \alpha_2) \), and a \( \Sigma_2 \)-specification \( S_2 = (G(S_2), S_2(\Pi_2)) \) called the metamodel of \( F \). An \( F \)-specification is a \( \Sigma_1 \)-specification \( S_1 = (G(S_1), S_1(\Pi_1)) \) which conforms to \( S_2 \).
DPF Methodology and Modeling Formalisms

- Modeling Formalism $F_2 = (\Sigma_1, M_2, \Sigma_2)$
- Meta-formalism $F_3 = (\Sigma_2, M_3, \Sigma_3)$
- $F_3$ is reflexive iff $(\Sigma_3 \sqsubseteq \Sigma_2)$
DPF Methodology and Modeling Formalisms

- Modeling Formalism $F_2 = (\Sigma_1, M_2, \Sigma_2)$
- Meta-formalism $F_3 = (\Sigma_2, M_3, \Sigma_3)$
- $F_3$ is reflexive iff $(\Sigma_3 \subseteq \Sigma_2)$
Ecore is the metamodel of the Eclipse Modeling Framework, EMF

- Meta-formalism $E_3$ (EClassifier, EReference), essentially Graph
- Modeling Formalism, $E_2$ (EPackage), typed over $E_3$:
  
  $t(EPackage) = t(EClassifier) = t(EClass) = t(EDataType) = EClassifier,$
  
  $t(eClassifiers) = t(eType) = t(eAttributes) = t(eReferences) = eStructuralFeatures$
Ecore is the metamodel of the Eclipse Modeling Framework, EMF

- **Meta-formalism** $E_3$ (EClassifier, EReference), essentially **Graph**
- **Modeling Formalism**, $E_2$ (EPackage), typed over $E_3$:
  
  \[
  t(EPackage) = t(EClassifier) = t(EClass) = t(EDataType) = \text{EClassifier}, \\
  t(eClassifiers) = t(eType) = t(eAttributes) = t(eReferences) = eStructuralFeatures
  \]
Ecore is the metamodel of the Eclipse Modeling Framework, EMF

- **Meta-formalism** $E_3$ (EClassifier, EReference), essentially **Graph**
- **Modeling Formalism** $E_2$ (EPackage), typed over $E_3$:
  \[ t(EPackage) = t(EClassifier) = t(EClass) = t(EDataType) = EClassifier, \]
  \[ t(eClassifiers) = t(eType) = t(eAttributes) = t(eReferences) = eStructuralFeatures \]
Outline

1. Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2. Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3. Applications
   - Metamodeling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
Tools implemented as plug-ins to the Eclipse platform

- Need tools for:
  - Drawing of models, partly finished June 2008
  - Serialization of models, work started
  - Validation of models, future research project
  - Model transformation tools, research started
  - Tools for version control of models
Built over Eclipse Graphical Modeling Framework, GMF
Will build separate tool for building signatures, conforms to a modeling language
Tool for creating models over signatures, i.e. modeling tool in the chosen language
Today drawing tool still some issues with diagrams with more than 3 nodes
Outline

1. Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2. Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3. Applications
   - Metamodeling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
Engineering techniques where models are first-class entities

- Evolved from the popularity of diagrammatic languages such as UML and ER and their popularity for specification and documentation of software systems
- Aims to raise the abstraction level of software development from text to models
- The development process is based on software models, not on plain text
Model Driven Architecture (MDA) approach from Object Management Group (OMG) where:

- First step is building platform independent domain models, PIM’s
- PIM’s are refined to platform specific models PSM’s
- Refinement process more or less automatized as model transformations
- Code generation and model refinement can be seen as model transformations
- Reuse of software models by using model transformations
MDD vs Traditional Development Processes

- **Traditionally**
  - Requirements
  - Analysis
  - System design
  - Implementation
  - Testing
  - Deployment

- **MDA**
  - Requirements
  - Analysis
  - System design
  - Implementation
  - Testing
  - Deployment

- **MDA**
  - Requirements
  - Analysis
  - System design
  - Implementation
  - Testing
  - Deployment

**PIM**: Platform Independent Model
**PSM**: Platform Specific Model

Yngve Lamo
Diagram Predicate Framework (DPF)
Models are no longer only for documentation

- Models should be **both**:
  - intuitive for **software engineers** to intercept and work with
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding
⇒ Challenge to combine intuition with formalization
Models are no longer only for documentation

- Models should be **both**:
  - intuitive for **software engineers** to intercept and work with
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding
⇒ Challenge to combine intuition with formalization
Models are no longer only for documentation

- Models should be **both**:  
  - intuitive for **software engineers** to intercept and work with 
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding  
⇒ Challenge to combine intuition with formalization
Some facts about UML

- UML is defacto industry standard for modeling
- UML can only express constraints on binary relations
- 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 modeling languages

- Nice semantics, proven abilities and several fundamental Software Engineering problems solved by use of formal methods
- Set based semantics (logic, algebras etc...)
- 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
Summary

- The industry have a lot of hidden knowledge, they have no need for formalization
- Modeling community work with fibrations, not index as we are trained to in mathematics
- Practical problems seems easy in theory but you really get dirt on your fingers
- We have obtained good understanding on modeling, meta modeling and model transformation
Further work

- Work on classification of model transformations, what properties has different class of transformations?
- Implement different formalisms in DPF (UML, XML, ...)
- Implement model transformation tool with code generation and cross platform translation
Outline

1. Introduction and Motivation
   - Background
   - Algebraic Specifications
   - Object Orientation

2. Diagram Predicate Framework
   - Diagram Predicate Framework (DPF)
   - Diagrammatic Specification
   - Instance

3. Applications
   - Metamodelling
   - Tools
   - Model-driven Engineering (MDE)
   - Model Transformations
Model Transformation Concepts

- Model Transformation = set of Transformation Rules + Coordinations (⇒ operational)
- Model Transaction = declarative definition of the transformation
- Transformation Engine = executes the rules
Transformation Rules

- Input and output patterns $P$ and $P'$
- Input and output instances $I$ and $I'$
- The models $M_1$ and $M_2$
- The matches $m$ and $[t](m)$
Transformation Rules

- Input and output patterns $P$ and $P'$
- Input and output instances $I$ and $I'$
- The models $M_1$ and $M_2$
- The matches $m$ and $[t](m)$

$G(M_1) \xleftarrow{\iota_P} I \xrightarrow{\iota} G(M_2)$

$P \xrightarrow{\iota_P} [=] \xrightarrow{m} \xrightarrow{[t](m)} G(M_2)$

$P' \xrightarrow{\iota_{P'}} [=] \xrightarrow{m} \xrightarrow{[t](m)} G(M_2)$

$G(M_1) \leftarrow I \xrightarrow{\text{trace}} I' \xrightarrow{\iota} G(M_2)$