Graph Transformation in SWE

D. Janssens

MoVES, June 1, 2007
Overview

- Graph Transformation - model of computation
- Applications in SWE
  - Visual language for refactoring
    - Extending the rules
    - Star grammars
- Model Driven Engineering
  - Model transformation
  - Traceability
- Delegation-based AOP
Graph transformation (double pushout)

Rule:

Step $G \cong H$:

Universiteit Antwerpen
GT as model of computation
program = set of rules

+ Visual, simple

Significant body of research: independence, concurrency

- Expressivity?

Syntax? Program structure?
Overview

• Graph Transformation - model of computation
• Applications in SWE
  • Visual language for refactoring
    ✓ Extending the rules
    ✓ Star grammars
• Model Driven Engineering
  ✓ Model transformation
  ✓ Traceability
• Delegation-based AOP
example refactoring: Push Down Method

- Push Down Method copies a method in a superclass to its subclasses
  - Variable number of subclasses
  - Variable structure of syntax tree (and access to variables, parameters and methods)
Cloning

- Cloning process replaces multiple nodes by
  - Singular nodes
  - New multiple nodes

- Notation $\frac{z}{(a, b)}$
  - For a multiple and b single nodes
  - Both a and b can be zero
Expanding graph variables

- Cloning all the *multiple nodes*
- Replace the variable nodes and their *arms* (incident edges) by the desired structures
Adaptive rules

- Rules allowing cloning and expansion
  - Clone variables represented by **multiple nodes** and a **cardinality**
  - Graph variables designate **variable nodes**, which are connected by **arms** to a set of **border nodes** (“stars”)
Results

- Cloning + expansion + attributes suffice for expressing most of Fowler’s refactorings
- Matching algorithms
- Eager vs. lazy cloning
- “shaped” expansion
- Other mechanisms: cloning subgraphs, cloning of variable nodes
- Proof-of-concept implementation (FUJABA, MOFLON)
Overview

- Graph Transformation - model of computation
- Applications in SWE
  - Visual language for refactoring
    - Extending the rules
    - Star grammars
- Model Driven Engineering
  - Model transformation
  - Traceability
- Delegation-based AOP
Star Grammars
Adaptive star rules

- Similar to star rules, but by allowing multiple nodes as border nodes of stars
Example: Adaptive Star Grammar

for method bodies
Decidability results

• Membership decidable?
• In general: no (Star Grammars can generate RE - string languages)
  – For a reasonably large class membership is decidable (“straight” stars, no chains)
  – Further research into parsing is necessary
Overview

- Graph Transformation - model of computation
- Applications in SWE
  - Visual language for refactoring
    - Extending the rules
    - Star grammars
- Model Driven Engineering
  - Model transformation
  - Traceability
- Delegation-based AOP
MDE

- Transformations - transformation engine
- Consistency management
- Traceability
Models - Diagrams (Meeting Scheduler)

Universiteit Antwerpen
All classes from the conceptual model should correspond to entities in the robustness model. Their attributes and attribute types should correspond. Both internal types and library types should be supported.
lookupStereotypes
{motmot.transprimitive=lookupStereotypes}

<<loop>>
For all classes in the CM
{motmot.transprimitive=For all classes}

<<each time>>
Is the Class related to an Entity?
{motmot.transprimitive=Is the Class related to an Entity?}

<<success>>
[not eachClassTracesToAnEntity()]
[eachClassTracesToAnEntity()]
true

<<failure>>
setFocus("Associate with an entity", new Object[] {classInCM});
Story Diagrams

```plaintext
lookupStereotypes
{motmot.transprimitive=lookupStereotypes}

<<loop>>
For all classes in the CM
{motmot.transprimitive=For all classes}

<<each time>>
[not eachClassTracesToAnEntity()]

<<success>>
[eachClassTracesToAnEntity()]

true

<<failure>>
Is the Class related to an Entity?
{motmot.transprimitive=Is the Class related to an Entity?}

<<code>>
setFocus("Associate with an entity", new Object[] {classInCM});
```
Story Pattern: “Is the Class related to an Entity?”

Model Query as a primitive graph transformation rule

Universiteit Antwerpen
Classes to Entities

Formalization by *triple graph grammars*

Universiteit Antwerpen
Classes to Entities

Formalization by *triple graph grammars*

*Universiteit Antwerpen*
Operational Rule Derivation

Higher Order Transformation produces:

Universiteit Antwerpen
Results

- Use of SDM, TGG for model transformation and inconsistency management - case studies
- Extensions (deep copy - interactivity)
- Transformation Contracts: design-by-contract instead of graph transformation
- MoTMoT transformation engine - MDA standards
Overview

- Graph Transformation - model of computation
- Applications in SWE
  - Visual language for refactoring
    ✓ Extending the rules
    ✓ Star grammars
- Model Driven Engineering
  ✓ Model transformation
  ✓ Traceability
- Delegation-based AOP  - Hans Schippers