Static Aspect Weaving with Graph Rewriting

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://www-st.inf.tu-dresden.de
Version 06-0.9, Jun 27, 2006
Overview

► Aspect Oriented Development (AOSD) and -Programming (AOP)
► Graph Rewrite Systems (GRS)
► Categories of GRS-based Weaving
► Generation of Aspect Weavers
► Conclusion
Aspect-Oriented Development

- Motivation: Separation of Concerns
  - a new kind of modularization
  - separation of cross-cutting code parts
- Technique: Integration by (Static) Weaving

Examples for Aspects:
- Synchronization
- Communication
- Instrumentalization
- Memory Management
Aspect-Oriented Development

Weaver-Tool

Algorithm

Debugging aspect

Persistence aspect

Debugging aspect

Persistence aspect

Debugging aspect
**Problem**

AOSD/AOP aims at different problem domains...

... weaving requires different specification languages.

A new weaver for every weave scenario!

Weavers are compilers...

Weaving can become complicated...
Idea

1) We need a uniform and formal technique to classify and specify AOP weavers.

2) Programs are colored graphs (abstract syntax graphs)...

Describe aspect weaving as graph-rewriting on those graphs.
Classes of AOP Systems

- Script-based AOP (e.g. RG, AspectJ, InjectJ)
  - aspects are modification rules

- Language-based AOP (e.g. D, AML)
  - aspects are specialized languages

- Graph-rewriting-based AOP
  - rewriting rules combine aspect fragments
Component + Aspect Graphs
GRS - Basics

Rewrite Rule

Host Graph

Redex

Derivation
AOP as GRS

Join Point $\rightarrow$ Redex in Component Graph
Aspect Fragment $\rightarrow$ Redex in Aspect Graph
Aspect Composer $\rightarrow$ Graph Rewrite Rule
Weave Operation $\rightarrow$ Direct Derivation
Weaver $\rightarrow$ Graph Rewrite System with
- a set of aspect composers,
- a component graph, and
- a set of aspect graphs (context-sensitive rules).
Example

Task: prepend statements to method entries

class: ClassDeclaration (name=Name)
  methods
    method: MethodDeclaration

aspectStatement: Statement (place=name)

class: ClassDeclaration (name=Name)
  methods
    method: MethodDeclaration

aspectStatement: Statement (place=name)

entry
Example (2)
Benefits

- Handling of all kinds of aspects possible
  - all we need is abstract syntax
- Uniform specification allows a classification of aspect weaving systems
- Certain classes of rewrite systems guarantee
  - termination
  - confluence (= deterministic results)
Category I: Aspect-Relating Rules

- Edge-addition rewrite system (EARS)
  - always congruent (= terminating + confluent)
  - weaving operation becomes a function
- Ideal for simple property aspects
  - e.g. persistency, synchronization, ...
Category II: Aspect-Additive Rules

- if it is an eXhaustive Graph Rewrite System (XGRS) and does not modify the redex
  - always congruent (= terminating + confluent)
- Ideal for orthogonal aspect code
  - e.g. Adaptive Programming
Category III: Component-Modifying Rules

- Exchange parts of components.
- Confluence and termination are not guaranteed.
- Indeterminism is acceptable if all normal forms are semantically equivalent.
Special Category: Aspects in Rules

- Aspect fragments are part of the right-hand sides.
- Similar to script-based AOP.
- Ideal for aspects with finite variability (because of finite set of rules).
Special Category: Component-Modifying Rules

- Intra-component rules
  - rewrite the component graph only
  - resemble standard code motion optimizations
- Ideal for optimizing aspect weavers.
  - e.g. RG (Reverse Graphics of Xerox)
## Comparison

<table>
<thead>
<tr>
<th>System</th>
<th>terminating</th>
<th>deterministic</th>
<th>aspect graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect Fragment Matching</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aspect-relating</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Aspect-additive</td>
<td>if exhaustive</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Component-modifying</td>
<td>if exhaustive</td>
<td>usually not</td>
<td>yes</td>
</tr>
<tr>
<td>Aspects in Rules</td>
<td>depends</td>
<td>depends</td>
<td>no</td>
</tr>
<tr>
<td>Intra-Component</td>
<td>depends</td>
<td>depends</td>
<td>no</td>
</tr>
</tbody>
</table>
Aspect Weaving is similar to Program Optimization

- Graph rewriting can express many program optimizations uniformly [Aßm96].
- Optimizations transform programs.
- Weavers transform programs.

So:

Graph rewriting can express many aspect weavings uniformly.
Generating Tools from Rewrite Specification

[Alexander Christoph, PhD 2004, University of Karlsruhe]
Conclusion for Aspect People

- GRS provide a **uniform** and **formal** way to specify and classify aspect weavings.
- Tool support for weavers.

- Open question:
  How much of AOP can be covered by this approach?
Conclusion for Compiler People

- Static Weavers are compilers.
  - AOP needs more formal background
  - Architectural languages are subsumed
- Compilers have a formal background
  - Both fields should know from each other!
- Compilers ARE simple Weavers
  - they crosscut declarations into statements
- Aspect Orientation will come after OOP.
  - A new kind of modularization
  - A new kind of software composition
  - Throw away OO!
The End

- Several slides are courtesy to Dr. Andreas Ludwig.