AspectJ and AP

Karl Lieberherr

Two enhancements

- A new kind of static crosscutting
  - `declare traversal t: strategy;`
    - introduces a new traversal method that traverses objects of Source(strategy) according to the strategy
- A new kind of type patterns
  - Define a set of types using a strategy; apply strategy to class graph.
    - `ContainedIn(strategy)`
    - `ReachableThrough(strategy)`

A new kind of static crosscutting

- `declare traversal t: strategy;`
  - introduces a new traversal method that traverses objects of Source(strategy) according to the strategy
  - this generalizes the AspectJ introduction `Type TypePattern.Id(Formals){Body}` that defines a new method on all types in TypePattern. With the traversal introduction the body depends on the current type.

Use of traversals

- The declared traversals can be invoked on objects o by giving a visitor object as argument: o.t(v).
- This is similar to DemeterJ.

Type patterns

- What is already in AspectJ:
  - A type pattern defines a collection of types.
  - Type names are type patterns.
  - A special type name: * picks out all types, including primitive types.
  - Subtype pattern: A+ denotes all subtypes of A.

New kinds of type patterns: Strategy type patterns

- define a set of types declaratively
- two kinds of strategy type patterns
  - ContainedIn(D) selects the set of all types that are in the scope of strategy D.
  - ReachableThrough(D) selects the set of all types t from which the target of D can be reached through the source of D and following D.

Scope of a strategy

- The scope of a strategy modulo a class graph is the set of types whose instances may appear on traversal paths of objects of the class graph.

Strategy type patterns

- Reachable(D) = {x| there is a path in the class graph from x to a node in ContainedIn(D)}
- ReachableThrough(D) = {x|ContainedIn(join(x to Source(D), D)) is not empty.

Examples

- For security purposes we want to advise all calls of get methods of classes from Portfolio to Posting. Strategy type pattern:

  ContainedIn(from Portfolio to Posting)

Examples

- We want to advise all methods of types from which Money is reachable through Retirement:

  ReachableThrough(from Retirement to Money)
- Reachable(Money): all types from which we can reach Money.

Traversal strategies

- D ::= [A,B] | join(D1,D2) | merge(D1,D2)
- We can use them in three different graphs relevant to programming:
  - call trees
  - class graphs
  - object trees

Interpretation of traversal strategies

- D ::= [A,B] | join(D1,D2) | merge(D1,D2)
- Source([A,B]) = A
- Target([A,B]) = B
- Source(join(D1,D2))=Source(D1)
- Target(join(D1,D2))=Target(D2)
- Source(merge(D1,D2))=Source(D1)
- Target(merge(D1,D2))=Target(D1)

Interpretation of traversal strategies

- D ::= [A,B] | join(D1,D2) | merge(D1,D2)
- WF([A,B]) = true // well-formed
- WF(join(D1,D2))=WF(D1) && WF(D2) && Target(D1) = Source(D2)
- WF(merge(D1,D2))= WF(D1) && WF(D2) && Source(D1)=Source(D2) && Target(D1)=Target(D2)

Dynamic call tree

- nodes are operation calls: labeled by operation name and arguments
- edges: a operation calls another operation
- Path back: contents of run-time stack

Interpretation of traversal strategies

- D ::= [A,B] | join(D1,D2) | merge(D1,D2)
- A and B are operation names
- [A,B]: the set of B-nodes reachable from A-nodes
- join(D1,D2): the set of Target(D2)-nodes reachable from Source(D1)-nodes following D1 and then following D2.

Interpretation of traversal strategies

- merge(D1,D2): the union of the set of Target(D1)-nodes reachable from Source(D1)-nodes following D1 and the set of Target(D2)-nodes reachable from Source(D2)-nodes following D2.

Translation Rules

- D1
- from A to B
- merge(D1,D2)
- join(D1,D2)

- t(D1)
- flow(A) && B
- t(D1) || t(D2)
- flow(t(D1)) && t(D2)

Source, Target definitions:
Source(from A to B) = A
Target(from A to B) = B
Source(join(D1,D2) = Source(D1)
Target(join(D1,D2) = Target(D2)
Source(merge(D1,D2)) = Source(D1)
Target(merge(D1,D2)) = Target(D1)

rules:
join: Target(D1) = Source(D2)
merge: Source(D1) = Source(D2)
Target(D1) = Target(D2)

Correspondences

• D1	• t(D1)
• from A to B	• flow(A) && B
• from A to *	• flow(A)
• from A via B to C	• flow(flow(A) && B) && C
• from A via B via C to E	• flow(flow(flow(A) && B) && C) && E
• merge(from A via B1 to C, from A via B2 to C)	• (flow(flow(A) && B1) && C) || (flow(flow(A) && B2) && C)
• merge(D1,D2)	• t(D1) || t(D2)
• join(D1,D2)	• flow(t(D1)) && t(D2)
• join (from A to B, from B to C)	• flow(flow(A) && B) && (flow(B) && C)
	• = flow(flow(A) && B) && C

subset(flow(B)) && flow(B) = subset(flow(B))

Class graph

- D
- [A,B]
- join(D1,D2)
- merge(D1,D2)

- PathSet(D)
- Paths(A,B)
- PathSet(D1).PathSet(D2)
- PathSet(D1) || PathSet(D2)

we are only interested in the set of nodes touched by the path sets -> subgraph of class graph

Object tree

- O
- [A,B]

- subgraph of O
- subgraph of O consisting of all paths from an A-node to a B-node.

Object tree

- O
- join(D1,D2)

- subgraph of O
- subgraph of O consisting of all paths following D1 concatenated with all paths following D2.

Object tree

- O
- merge(D1,D2)

- subgraph of O
- subgraph of O consisting of all paths following D1 or following D2.

does not use prematurely terminated paths

Our Body

NervousSystem = CentralNervousSystem
PeripheralNervousSystem.

CentralNervousSystem = Brain SpinalCord.

PeripheralNervousSystem = SensoryDivision
MotorDivision.

MotorDivision = SomaticNervousSystem
AutonomicNervousSystem.

Our Body

SomaticNervousSystem = “voluntary action”

AutonomicNervousSystem = “involuntary action”
  SympatheticNervousSystem
  ParasympatheticNervousSystem.

SympatheticNervousSystem = “fight or flight”.

ParasympatheticNervousSystem = “rest and digest”.