Discussion with Gregor Kiczales at UBC

- **Ontology of AOP**
- **Ontology is the study of what there is, an inventory of what exists. An ontological commitment is a commitment to an existence claim for certain entities.**
- **Slides 2 - 5 and the last one are Gregor's**

An ontology is, in simple terms, a collection of concepts with relations among them plus constraints on the relations.

basis of crosscutting

- **a *join point model* (JPM) has 3 critical elements**
  - what are the join points
  - means of identifying join points
  - means of specifying semantics at join points

basis of crosscutting

- **a *join point model* (JPM) has 3 critical elements**
  - what are the join points
    - in AspectJ
      - points in runtime call graph
      - class members
  - means of identifying join points
    - in AspectJ
      - pointcuts
      - member signatures (plus …)
  - means of specifying semantics at join points
    - in AspectJ
      - advice
      - define members

dynamic JPM
static JPM

range of AOP languages

JPM	join points	means of … join points	
		identifying	specifying semantics at
AspectJ dynamic JPM	points in execution call, get, set…	signatures w/ wildcards & other properties of JPs	advice
static JPM	class members	signatures	add members
Composition Filters	message sends & receptions	signature & property based object queries	wrappers declarative (filters) imperative (advice)
Hyper/J	members	signatures	add, compose (and remove) members
Demeter traversals	when traversal reaches object or edge	class & edge names	define visit method

range of AOP languages

JPM	join points	means of … join points	
		identifying	specifying semantics at
AspectJ dynamic JPM	points in execution call, get, set…	signatures w/ wildcards & other properties of JPs	advice
static JPM	class members	signatures	add members
Composition Filters	message sends & receptions	signature & property based object queries	wrappers declarative (filters) imperative (advice)
Hyper/J	members	signatures	add, compose (and remove) members
Demeter traversals	types	succinct traversal specs	generate traversals

See next slide for changes to Demeter

range of AOP languages

JPM	join points	means of … join points: identifying	means of … join points: specifying semantics at
AspectJ dynamic JPM	points in execution call, get, set…	signatures w/ wildcards & other properties of JPs	advice
static JPM	class members	signatures	add members
DemeterJ, Demeter/C++ dynamic JPM	when traversal reaches object or edge	visitor method signatures	visitor method bodies
static JPM 1	class members	traversal spec. s class graph g	s + g (result = traversal implementation)
static JPM 2	class members	class names	add members
static JPM 3	class members	class graph	class graph with tokens=grammar (result = parsing and printing implementation)

range of AOP languages

JPM	join points	means of … join points: identifying	means of … join points: specifying semantics at
AspectJ dynamic JPM	points in execution call, get, set…	signatures w/ wildcards & other properties of JPs	advice
static JPM	class members	signatures	add members
DJ dynamic JPM 1	when traversal reaches object or edge (method traverse)	visitor method signatures	visitor method bodies
dynamic JPM 2	when traversal reaches object (methods fetch, gather, asList)	source and targets of traversal	method name (fetch, gather, asList)
dynamic JPM 3	nodes in object graphs	trav. spec. s class graph g object graph o	s+g+o(result = traversal implementation = edges to traverse at nodes in object graph)

Composing join point models

- **Traversal Spec JPM: In Demeter we use traversal specifications and the class graph to define a traversal implementation (either static or dynamic)**
- **Visitor JPM: The result of Traversal Spec. JPM is used to define a second JPM:**
  - The traversal implementation defines nodes and edge visits.
  - Visitor signatures define the nodes and edges where additional advice is needed: they are the means of identifying join points.
  - The means of specifying semantics at join points are the visitor bodies.

DJ: dynamic JPM 3

- **The join points are nodes in object graphs. They are not dynamic call graph join points nor class members!**
- **The means of identifying the join points for a given object graph o are a strategy s and the class graph g. o must conform to g.**
- **The means of specifying the semantics at the join points are again s and g. See paper with Mitch Wand for the formal details behind this JPM.**

DemeterJ: static JPM 1

- **The means of identifying the join points and of specifying the semantics at the join points are the same.**
- **The reason is that s+g both**
  - select the classes that will get traversal semantics
  - determine the details of the traversal semantics

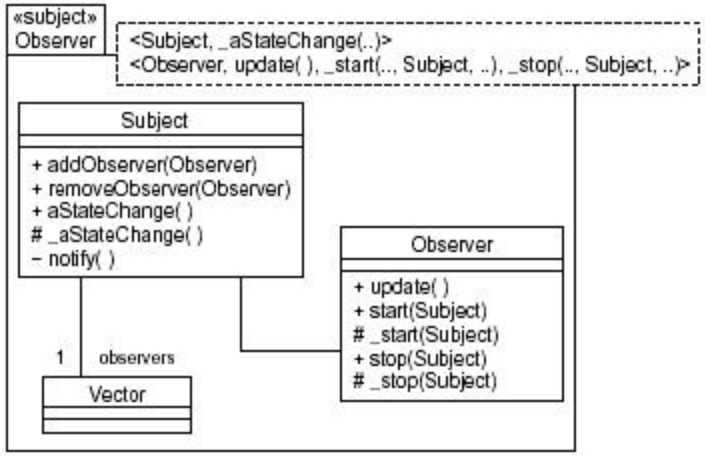
DemeterJ: static JPM 3

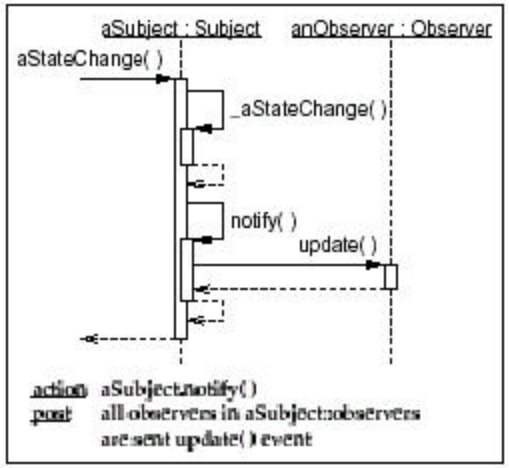
- **The means of identifying the join points (class members) is done by the class graph.**
- **When we add tokens to the class graph we get a grammar that contains instructions for parsing and printing behavior.**
- **A grammar is an aspect (external representation aspect): the adhoc implementation cuts across all classes.**

AO design in UML

[Clarke, Walker]

- **Composition Patterns**
  - static JPM (~ Hyper/J)
  - binds pattern to base code
  - dynamic JPM (~ AspectJ)
  - what happens in pattern





- **UML class & interaction diagrams already crosscut**
  - by-class vs. by-interaction organizations