Early Aspects in Agent Oriented Modeling

Rushikesh K Joshi
Dept. of CSE
IIT Bombay
Plan of the talk

- Introduction to ideas from aspect orientation
- Applying aspect orientation at requirements level
- Aspect oriented paradigms in agent oriented methods
A Canvas of Programming Abstractions

events  types  D-structures
variables  structures  functions  exceptions
classes  objects  connectors  continuations
components  processes  packages
threads  agents  ambients  files
synchronizers
Abstractions + Related Processes
Abstractions + Related Processes + Properties
Is this space enough for today’s computations?

- Maybe enough but ...
- Do we have a clean organized view of all aspects of your software that is traceable from architecture to implementations?
- Do you maximize reuse?
  - Could you eliminate all redundancies!
The key: methods of separation and integration
Let’s Take a look at Some Empirical Studies

- Code redundancies reported (an old research)
  - Application projects: 75%
  - System programs: 50%
  - Telecommunication projects: 70%

- Reengineering projects find redundancies and eliminate them: 20-50%

- A latest study: 60% code in one Java class library was found to be redundant
How to eliminate the redundancies?

- Just keep a copy of the redundant code and simply use it as a black box through conventional techniques.
  - Not always possible!
    - Technology imposes limitations
      - methods and models of structuring
      - varying flexibility for reaching meta levels
  - We can trace the problem to mixup of multiple concerns
Another perspective on non-separated concerns

- Redundancy results when a concern occurs in many entities, but each manifests it independently.
- A single bundle may also host multiple concerns that are tangled and not separated.
- A concern may get scattered over many entities.
- Some examples follow.
Concerns that tangle with other concerns

To tangle: To mix together or intertwine in a confused mass

- Functional code (business logic) and properties about the code
  - Assertions that capture contracts (pre/post/invariants)
  - Invariants across objects
  - Creational control and object’s instance behavior
- Exception handling code and functional code
- Nonfunctional code and functional code
  - Whenever function pop() is invoked, print the return value to a file
  - Log all calls to a specific object
  - Log all calls to all objects
  - Make a distributed object persistent
Programming paradigms influence the way we organize software...

- The problem can be attacked at programming level
  - By evolving programming paradigms
Separation of concerns at requirements level

- Separately express the requirements concerns
- Can you change them independently?
  - Or does a change in one use case lead to changes in many other use cases?
- Are requirements specs tangled?
  - The question is not about correctness and completeness
  - It is about modularity in expression in requirements capture
An Agent Oriented Requirements Capture Model (entities)

- **agent**
- **position**
- **role**

**ACTORS**

- **goal**
  - Desires/intentions
  - Not defined precisely
  - To be fulfilled

- **softgoal**

- **plan**
  - Set of actions for satisfying goal(s)

- **resource**
  - A resource in the system
An Agent Oriented Requirements Capture Model (relations)

- **goal**
  - dependency
  - contribution

- **goal**
  - +
  - goal

- **goal**
  - +
  - softgoal

- **goal**
  - Decomposition (AND/OR)

- **agent**
  - occupies
  - pos.
  - covers
  - role

- **dependent**
- **dependee**

- **Occupation And coverage**
An example entity model

With appropriate AND/OR constraints

A goal analysis
Some Extensions for Separation of Concerns

- Extended actors
  - Before
  - After
- Abstract actors
  - Before
  - After
- Similar extensions for goals
- Shared goals
Aspect Goals

- Goals may be decomposed further into subgoals, and shared goals.
- But it is not always possible to share goals “as it is”.
- Certain refinements may be necessary.
- Aspect goals: an example
Meta goals

- Supporting meta goals (like around advise, before advise and after advise)
  - Wrapper goals (performance criteria)
  - pre goals (preconditions)
  - Post goals (postconditions)
Goal Ordering

- Partial orders may be defined
- Does not indicate goal decomposition but captures workflows (activity dependencies in UML)
- An example
Early Aspects: some pointers

- Concerns
  - Core Functionality
  - Security
  - Deadlines
  - Persistence
  - Mobility
  - Replication

- Tangling within the specs
Aspect Oriented Programming Constructs:
A Summary, More Examples and Related Approaches
Join Points

- A point in a source program
  - Method call
  - Constructor call
  - Variable read/write
  - Exception handler
  - Variable initializer
  - Destructor
Point cuts

- A set of join points + optionally some of the execution context values
  - Call (void Point.setX(int))
    - A call to a specific function
  - Call (public * Figure.*(..))
    - Calls to all public functions on Figure
  - Pointcut move: call ... || call ...
    - Any of the above calls
  - !instanceof (X) && call ...
    - Call originates not from instance of X and to specified method
**Advices**

- Advices executed at the code at joinpoints for given pointcuts
  - Before advice
    - After reaching a join point, but before the computation proceeds
  - After advice
    - After the computation at join point has completed
  - Around advice
    - Run first. Proceed() inside around advice makes the computation proceed
  - After returning
  - After throwing

- Introductions: add new fields to classes, change relationships
Some more examples

- Aspects in a distributed objects domain
  - Object’s functionality
  - It’s location
  - It’s itinerary
  - Communication and synchronization
  - Its persistence
  - Its security
Aspect Orientation in middleware

- Write objects in your application first
- Add on services to the application later
  - Use AOP techniques (interceptor/static transformation) techniques
Feature interaction problem

- Effects of one aspect may interfere with that of another

- Careful ordering of aspect application is important
Product Line Approaches

High level transformation code

+ 

High level Base code

→ Actual Variant
Base-Meta Separation

- Meta-object protocols
- Reflection
  - Ideas are quite old
  - Some of the recent technologies have discovered them only now!
Filter Objects Approach

- Message based paradigm
- Based on interfaces and capture on messages
- Dynamic and First class aspects
- Pluggability at runtime
- Weaving not possible
- Filter objects for C++/Java/CORBA/COM, patterns, configurations
Open Problems

- Static vs. Dynamic aspects
  - Commercial Tools and Technologies are picking up
- Early aspects and traceability into code
- Aspects in processes
- Large scale applications and actual practice
- Impact on Systems design and software Engineering lifecycle in general
- Impact on Modeling Languages