Lecture #13 (F13)
27 April 2006

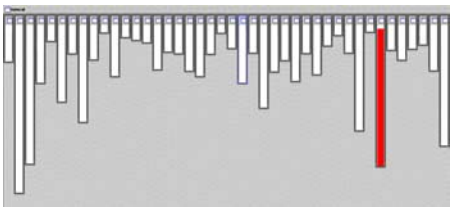
- Aspect-oriented programming (AOP)
- Aspect-oriented modelling (AOM)
- Introduction to agents
- Modelling agents and mappings from SOA to agent technology

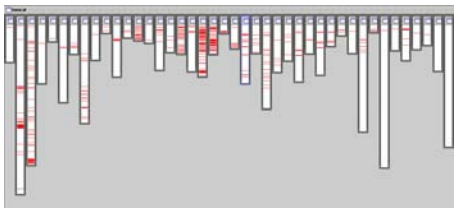
Aspect-oriented programming (AOP)

Based on material by

Mik Kersten
Palo Alto Research Center

Modularisation





- Good modularity
- XML parsing in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in one box
- Logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

Cost of tangled code

- Redundant code
  - same fragment of code in many places
- Difficult to reason about
  - non-explicit structure
  - the big picture of the tangling isn't clear
- Difficult to change
  - have to find all the code involved
  - and be sure to change it consistently
  - get no help from OO tools

The AOP idea

- Crosscutting is inherent in complex systems
- Crosscutting concerns
  - have a clear purpose
  - have a natural structure
    - defined set of methods, module boundary crossings, points of resource utilization, lines of dataflow…
- Capture the structure of crosscutting concerns explicitly...
  - in a modular way
  - with linguistic and tool support
- Aspects are
  - well-modularized crosscutting concerns

AspectJ

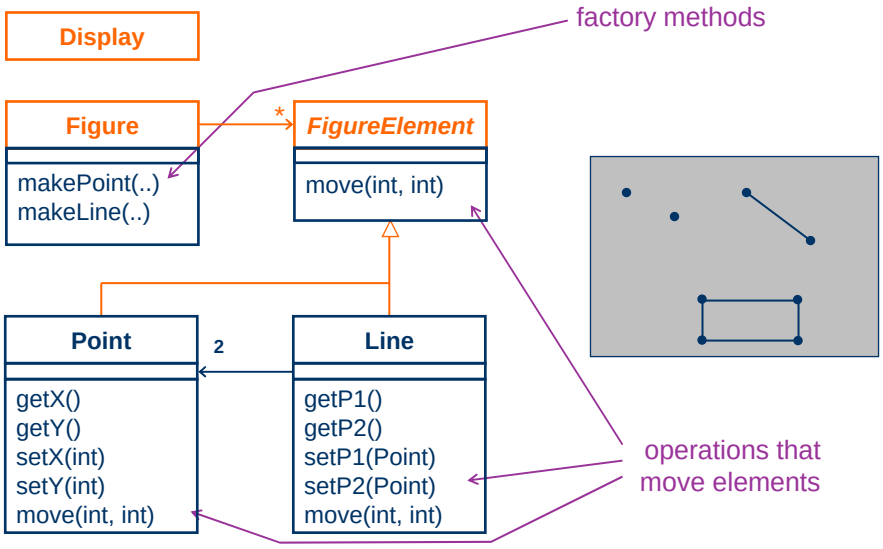
- AspectJ.org is a PARC project
  - partially funded by DARPA under contract F30602-97-C0246
- A small and well-integrated extension to Java
  - outputs .class files compatible with any JVM
  - all Java programs are AspectJ programs
- A general-purpose AO language
  - just as Java is a general-purpose OO language
- Includes IDE support
  - emacs, JBuilder, Forte 4J, Eclipse
- Freely available implementation
  - compiler is Open Source
- User feedback is driving language design
  - users@aspectj.org, support@aspectj.org

- Download the tools and docs at: http://aspectj.org
- Get the eclipse plug-in: http://eclipse.org/ajdt

Basic mechanisms

- 1 overlay onto Java
  - dynamic join points
    - “points in the execution” of Java programs
- 4 small additions to Java
  - pointcuts
    - pick out join points and values at those points
      - primitive, user-defined pointcuts
  - advice
    - additional action to take at join points in a pointcut
  - inter-class declarations (aka “open classes”)
  - aspect
    - a modular unit of crosscutting behavior
      - comprised of advice, inter-class, pointcut, field,constructor and method declarations

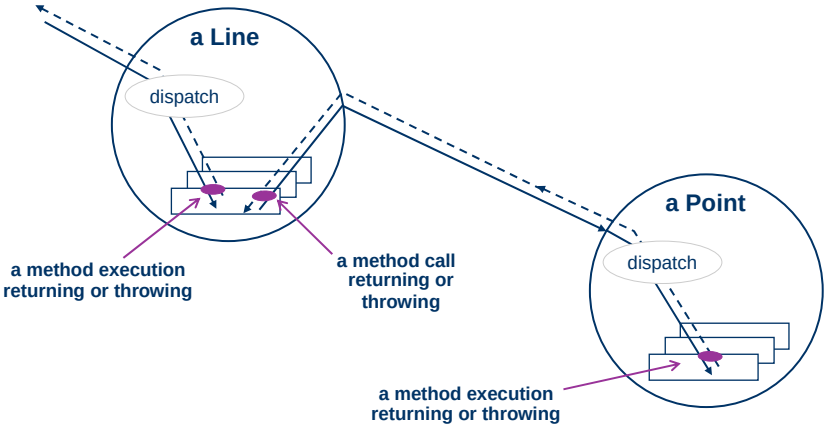
A simple figure editor



Join points

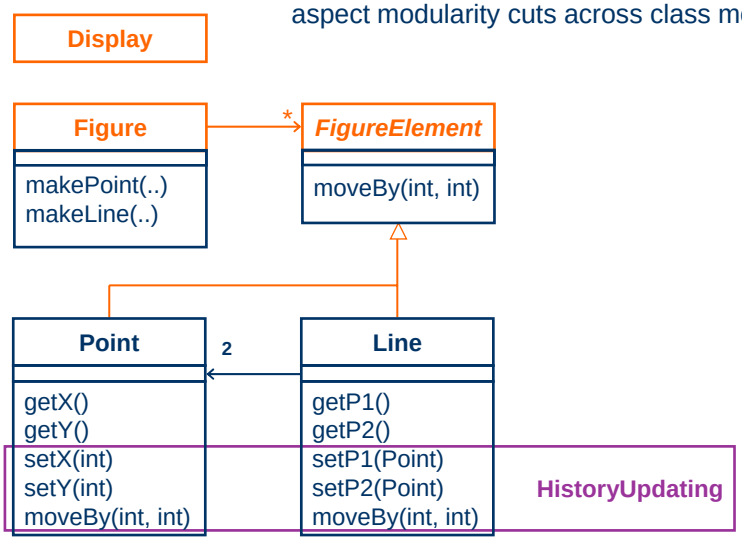
key points in dynamic call graph

imagine `l.move(2, 2)`



aspects crosscut classes

aspect modularity cuts across class modularity



Without AspectJ

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update(this);
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update(this);
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
    Display.update(this);
  }
  void setY(int y) {
    this.y = y;
    Display.update(this);
  }
}
```

- no locus of “display updating”
  - evolution is cumbersome
  - changes in all classes
  - have to track & change all callers

With AspectJ

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
  }
  void setP2(Point p2) {
    this.p2 = p2;
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
  }
  void setY(int y) {
    this.y = y;
  }
}
```

```
aspect DisplayUpdating {

  pointcut move(FigureElement figElt):
    target(figElt) &&
    (call(void FigureElement.moveBy(int, int) ||
     call(void Line.setP1(Point))             ||
     call(void Line.setP2(Point))             ||
     call(void Point.setX(int))               ||
     call(void Point.setY(int)));


  after(FigureElement fe) returning: move(fe) {
    Display.update(fe);
  }
}
```

- clear display updating module
  - all changes in single aspect
  - evolution is modular

Aspect-oriented modelling (AOM)

Based on material by
Robert France
Department of Computer Science, Colorado State University

Problem

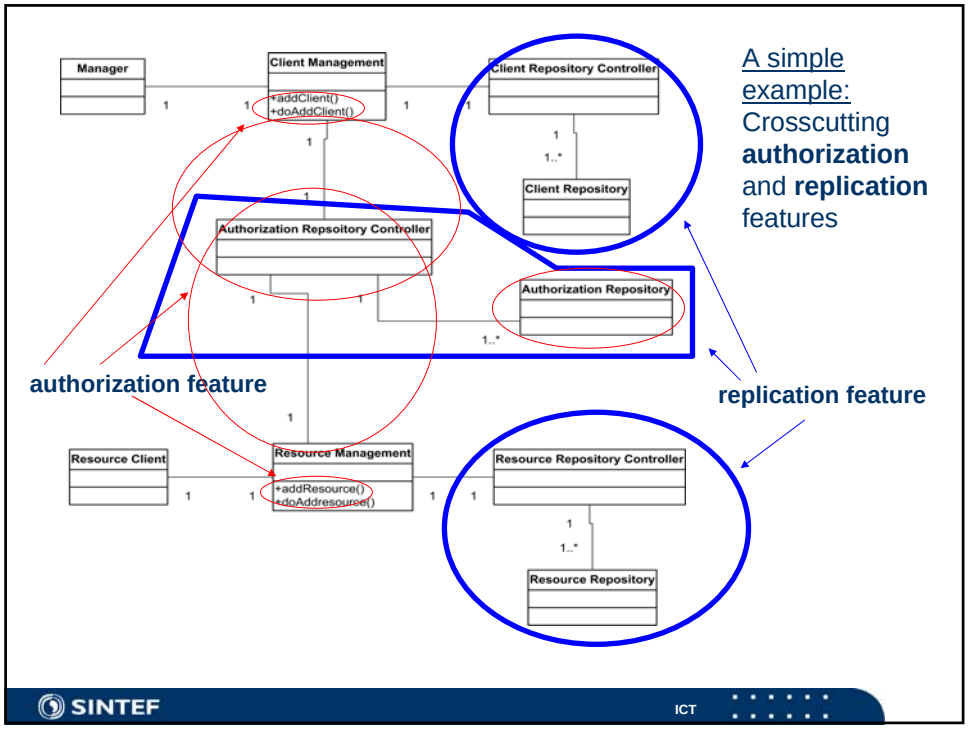
- How do we manage the complexity of analyzing and evolving intertwined dependability features?
- Through application of the separation of concerns principle …

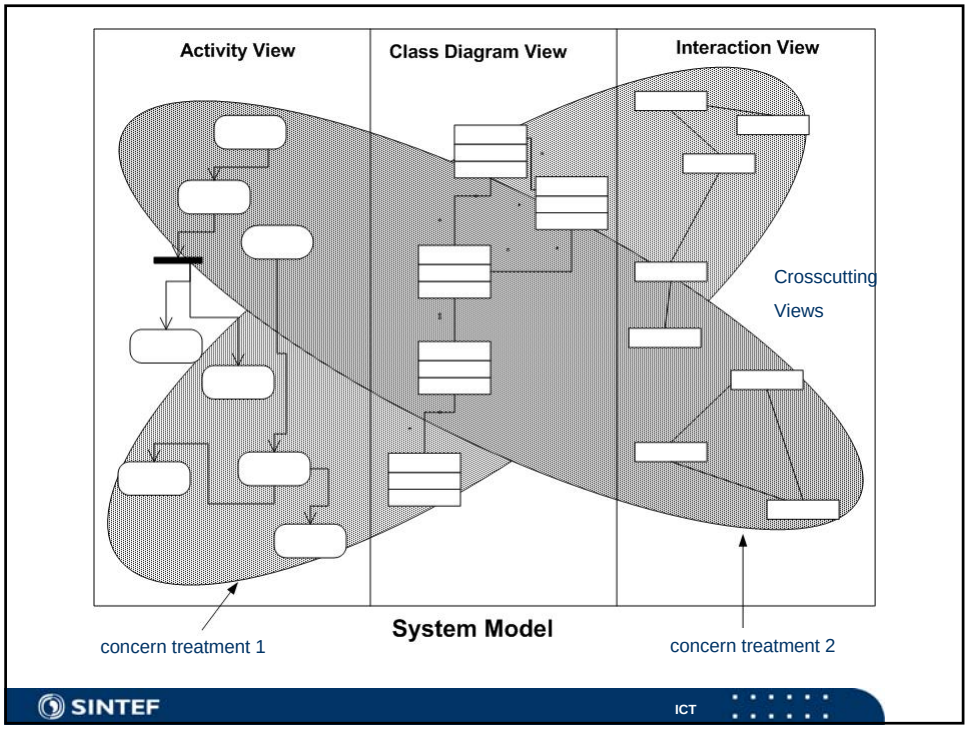
Motivation

- Designers of dependable systems must cope with multiple pervasive and competing concerns
  - Security
  - Safety
  - Fault tolerance
  - Functional concerns
- Developing a balanced design requires making trade-offs
  - Need to consider alternative ways of satisfying dependability goals

Multidimensional separation of concerns

- Current modelling techniques allow separation of concerns along a fixed set of dimensions
  - UML dimensions: static structural, state changes, activity flows, instance interactions
- A design modularization based on features that address a subset of design concerns may result in the distribution of other features across the design structure.


Manager
Client Management
+addClient()
+doAddClient()
Client Repository Controller
Client Repository
Authorization Repsoitory Controller
Authorization Repository
Resource Client
Resource Management
+addResource()
+doAddresource()
Resource Repository Controller
Resource Repository
A simple example: Crosscutting **authorization** and **replication** features
authorization feature
replication feature
SINTEF
ICT


Activity View
Class Diagram View
Interaction View
Crosscutting Views
concern treatment 1
System Model
concern treatment 2
SINTEF
ICT

The problem with cross-cutting features

- … understanding, analyzing and changing them!
  - Information is distributed
  - Maintaining consistency in the presence of changes is problematic
  - Difficult to consider alternative treatments
- Lack of attention to balancing dependability concerns early in the development cycle can lead to major re-architecting in later stages of development

A solution: Aspect-oriented modelling (AOM)

- Localize crosscutting features
  - Eases understanding of features
  - Eases evolution of features
  - Eases replacement of features with alternatives
- Crosscutting features can be isolated if distributed elements have common structural and behavioural features
  - Isolated features are described as solution patterns
- Aspect-oriented modelling allows developers to conceptualize, describe, analyze, and communicate crosscutting features separately from other treatments

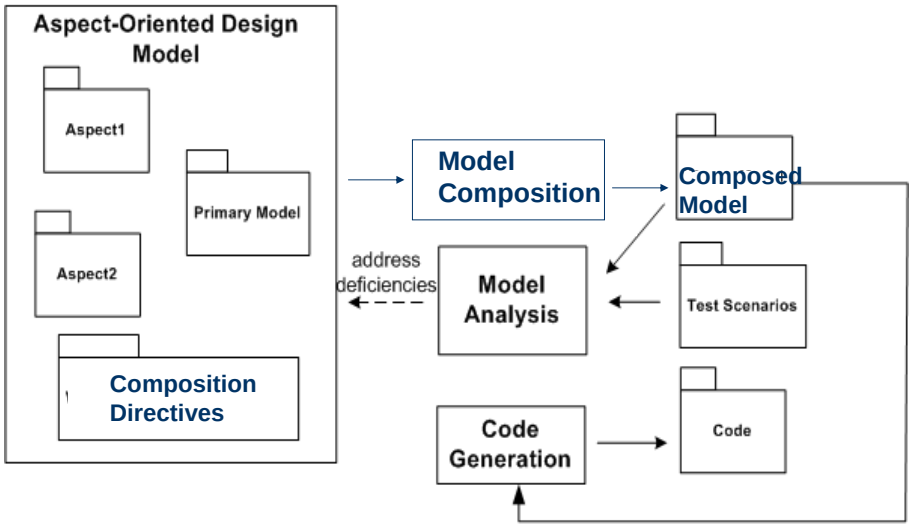
Key concepts

- A concern is a problem and a set of properties that determines acceptable solutions.
  - Concerns are addressed/treated in a design
  - Specification of concerns is out-of-scope of this talk
  - Focus is on specification of concern solutions
- A concern solution (or feature) is a description of a behavior that addresses the concern
- A feature that is distributed across the primary structure of a model is said to crosscut the model
- An aspect model is a description of a crosscutting concern solution.

Separating crosscutting concerns

- An AOM design model consists of
  - a primary model: reflects core design decisions
  - aspect models: each describes a solution for a concern not addressed in the primary model
  - composition directives: constrain how aspect models are composed with the primary model

A basic AOM approach



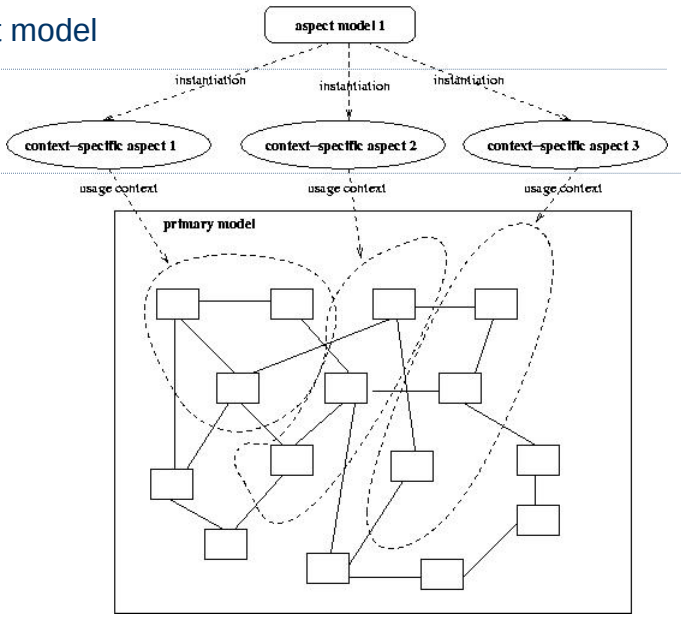
Aspect models

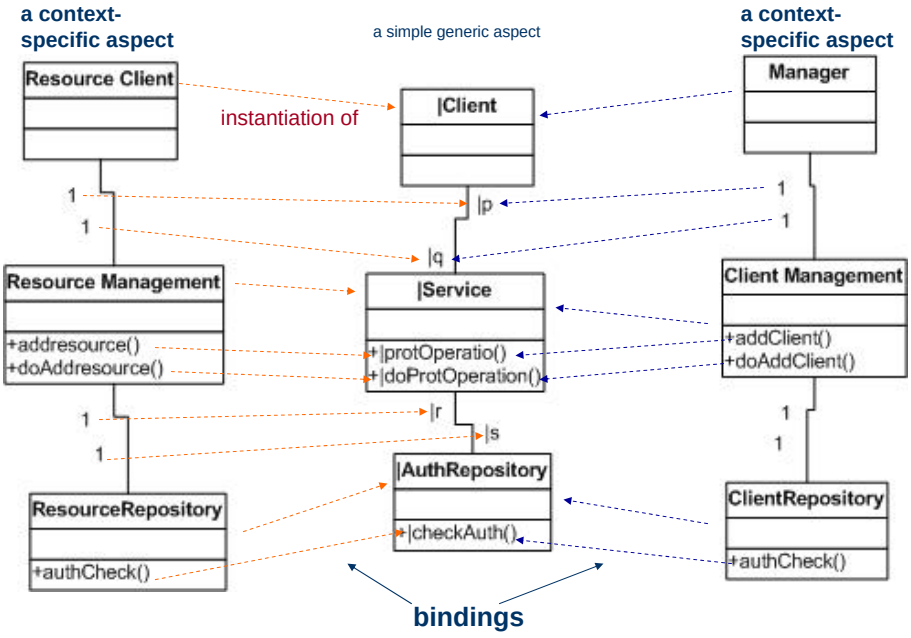
- Generic aspect model: A pattern of behavior
- Context-specific aspect model: An instantiation of a generic aspect model

(Generic) aspect model

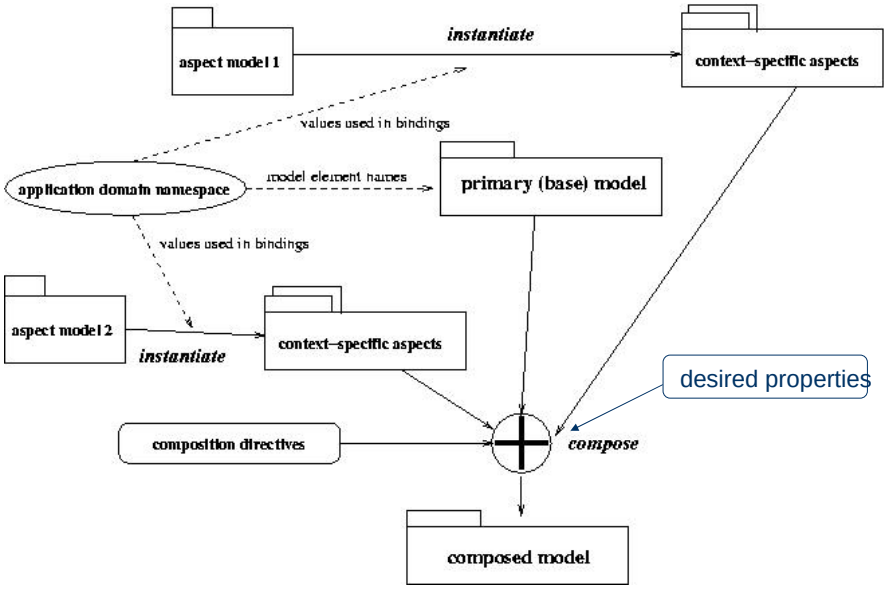
Context-specific aspect models

Primary model



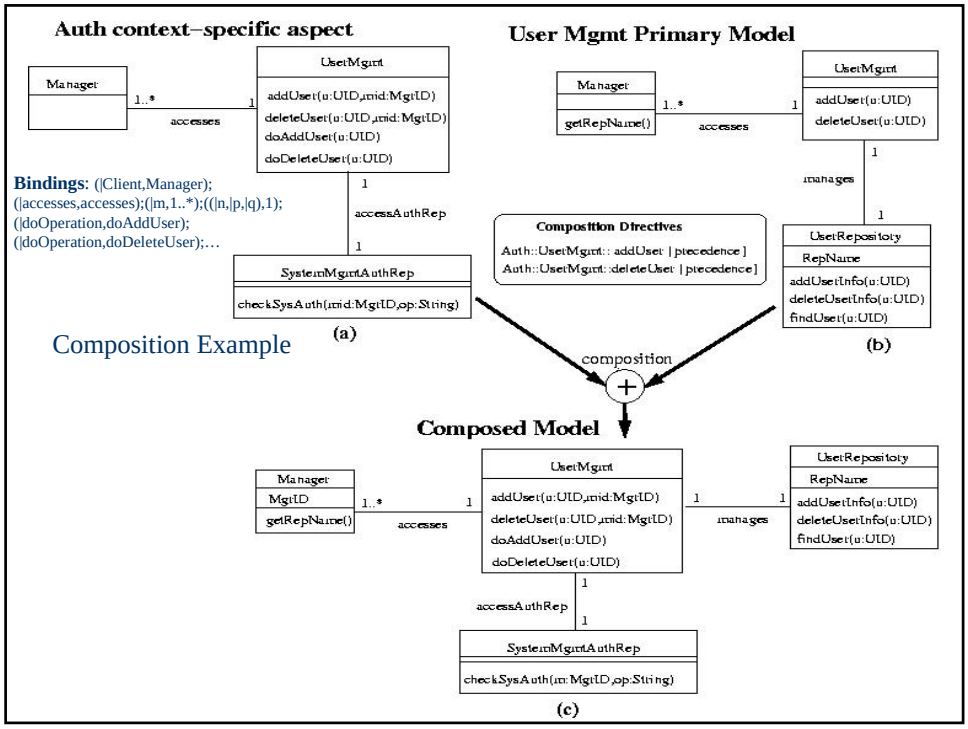


Overview of model composition



Composition process overview

- Composition involves
  1. Instantiating aspect models to produce context-specific aspect models
  2. Composing context-specific aspect and primary models
- Step 2 uses a basic name-based composition procedure
  - Elements in the context-specific aspect are merged with elements with the same names in the primary model
- Composition directives can be used to override the default name-based procedure

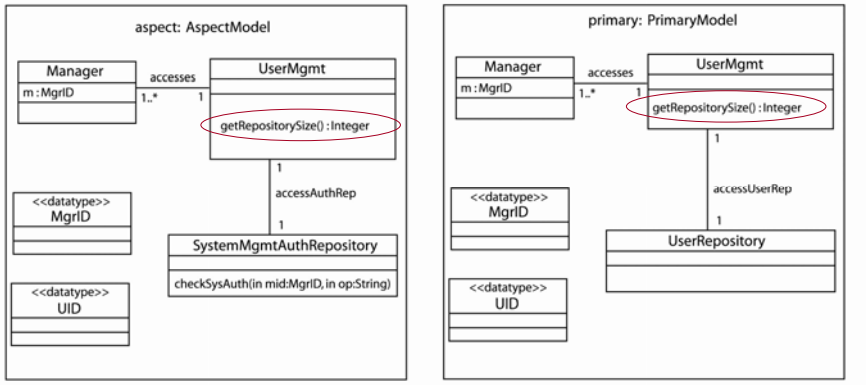


Composition Example

Composition Directives

- Composition directives can be used to vary how an aspect is composed with primary models
  - Two types: low-level and high-level directives
- Low-level directives can be used to transform aspect models, primary models, and intermediate forms of composed models so that composition produces models with desired properties:
  - specify precedence relationships between matching elements with conflicting properties
  - specify the deletion of model elements
  - specify the inclusion of new model element
  - rename model elements
  - replace references to model elements
- High-level directives are used to specify the order in which multiple aspects are composed with a primary model

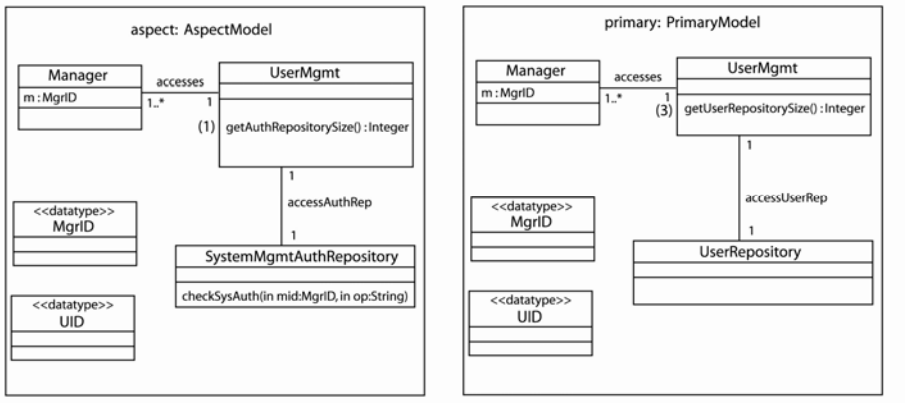
Resolving conflicts with directives



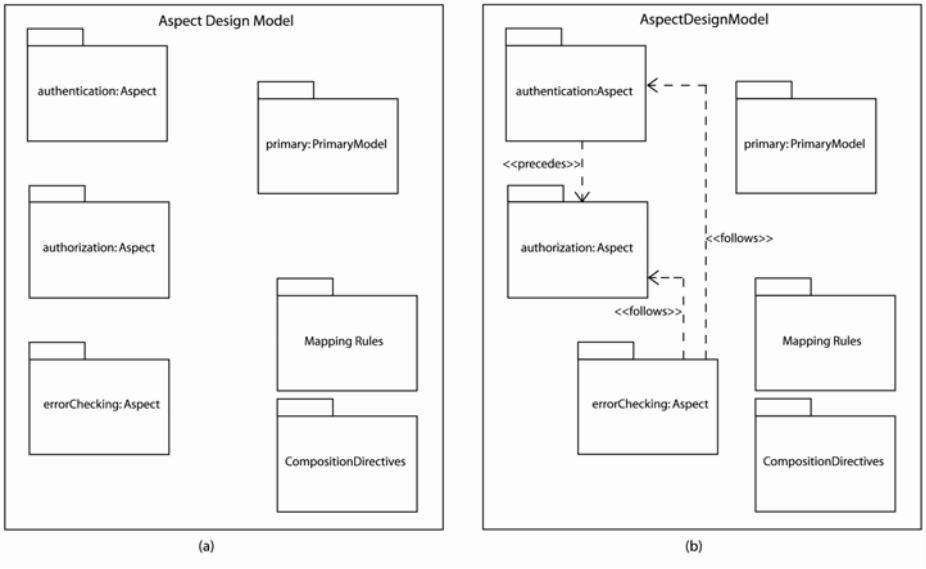
Conflict: *primary::UserMgmt::getRepositorySize()* returns the size of *UserRepository*,
aspect::UserMgmt::getRepositorySize() returns the size of *SystemMgmtAuthRepsoitry*.
Conflict resolution: use the rename directive to rename one or both operations, and the replaceReferences directive to update references to the old name.

(1) rename aspect::UserMgmt::getRepositorySize()
to aspect::UserMgmt::getAuthRepositorySize()
(2) replaceReferences
aspect::UserMgmt::getRepositorySize()
with aspect::UserMgmt::getAuthRepositorySize()
in aspect

(3) rename primary::UserMgmt::getRepositorySize()
to primary::UserMgmt::getUserRepositorySize()
(4) replaceReferences
primary::UserMgmt::getRepositorySize()
with primary::UserMgmt::getUserRepositorySize()
in primary



Example of high-level directives



AOM versus AOP

- Common theme: support for separation of crosscutting solutions
- Code versus models
  - Code provides a single view of behavior
    - AOP aspects crosscut source code modules
  - Models provide multiple overlapping views of system
    - Views support separation of concerns
    - AOM aspects crosscut views in a model
- AOM design models are not necessarily descriptions of aspect-oriented programs.
  - AOM design models can be transformed to non aspect-oriented programs

Introduction to agents

Based on material by
Martin L. Griss
School of Engineering, UC, Santa Cruz

Agents and agency

- What is an Agent?
  - There seem to be as many definitions as there are users of it
- Dictionary:

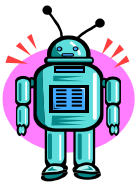
  a·gent *n.*

  1. One that acts or has the power or authority to act.
  2. One empowered to act for or represent another: *an author's agent; an insurance agent.*
  3. A means by which something is done or caused; instrument.
  4. A force or substance that causes a change: *a chemical agent; an infectious agent.*
  5. A representative or official of a government or administrative department of a government: *an FBI agent.*
  6. A spy.
  7. *Linguistics.* The noun or noun phrase that specifies the person through whom or the means by which an action is effected.

What about human agents?

- Describe some characteristics of human agents
  - carries out some task for someone else (master/slave)
  - is given goals but has considerable leeway on how to achieve these goals
  - probably moves around
  - observes features of her or his environment
  - interacts with, and perhaps changes to some extent, his or her environment of objects and (possibly) other agents
  - has a memory to store instructions, goals and other information
  - (hopefully) exhibits some intelligence
- Keywords
  - goals, observations, autonomy, intelligence, mobility

How about software agents?



- Artificial intelligence (AI) agents
  - First to use agent term in the computer world
  - Usually individual 'creatures' living in an environment where they interact in some way
  - Agents are more or less autonomous
  - Most AI agents do not interact with other agents in the same environment
  - Exhibit some kind of intelligence
  - A good example: robot interacting with a real world environment