# Lecture 16: Object Oriented Modeling Methods

## Basics of Object Oriented Analysis

- Notations used
- Modeling Process

## Variants

- Coad-Yourdon
- Shlaer-Mellor
- Fusion
- UML

## Advantages and Disadvantages

# Object Oriented Analysis

## Background

Model the requirements in terms of objects and the services they provide

Grew out of object oriented design

partitions the problem in a different way from structured approaches

Poor fit moving from Structured Analysis to Object Oriented Design

## Motivation

OOA is (claimed to be) more 'natural'

As a system evolves, the functions (processes) it performs tend to change, but the objects tend to remain unchanged…

…so a structured analysis model will get out of date, but an object oriented model will not…

…hence the claim that object-oriented systems are more maintainable

OOA emphasizes importance of well-defined interfaces between objects

compared to ambiguities of dataflow relationships

*NOTE: OO applies to requirements engineering because it is a modeling tool. But in RE we are modeling domain objects, not the design of the new system*

# Modeling primitives

## Objects

an entity that has state, attributes and services

Interested in problem-domain objects for requirements analysis

## Classes

Provide a way of grouping objects with similar attributes or services

Classes form an abstraction hierarchy though 'is_a' relationships

## Attributes

Together represent an object's state

May specify type, visibility and modifiability of each attribute

## Relationships

'is_a' classification relations

'part_of' assembly relationships

'associations' between classes

## Methods (services, functions)

These are the operations that all objects in a class can do…

…when called on to do so by other objects

E.g. Constructors/Destructors (if objects are created dynamically)

E.g. Set/Get (access to the object's state)

## Message Passing

How objects invoke services of other objects

## Use Cases/Scenarios

Sequences of message passing between objects

Represent specific interactions

# Key Principles

***See also:*** *van Vliet 1999, section 12.2*

## Classification (using inheritance)

Classes capture commonalities of a number of objects

- Each subclass inherits attributes and methods from its parent
- Forms an 'is_a' hierarchy

Child class may 'specialize' the parent class

- by adding additional attributes & methods
- by replacing an inherited attribute or method with another

Multiple inheritance is possible where a class is subclass of several different superclasses.

## Information Hiding

internal state of an object need not be visible to external viewers

Objects can encapsulate other objects, and keep their services internal

- useful for forming abstractions

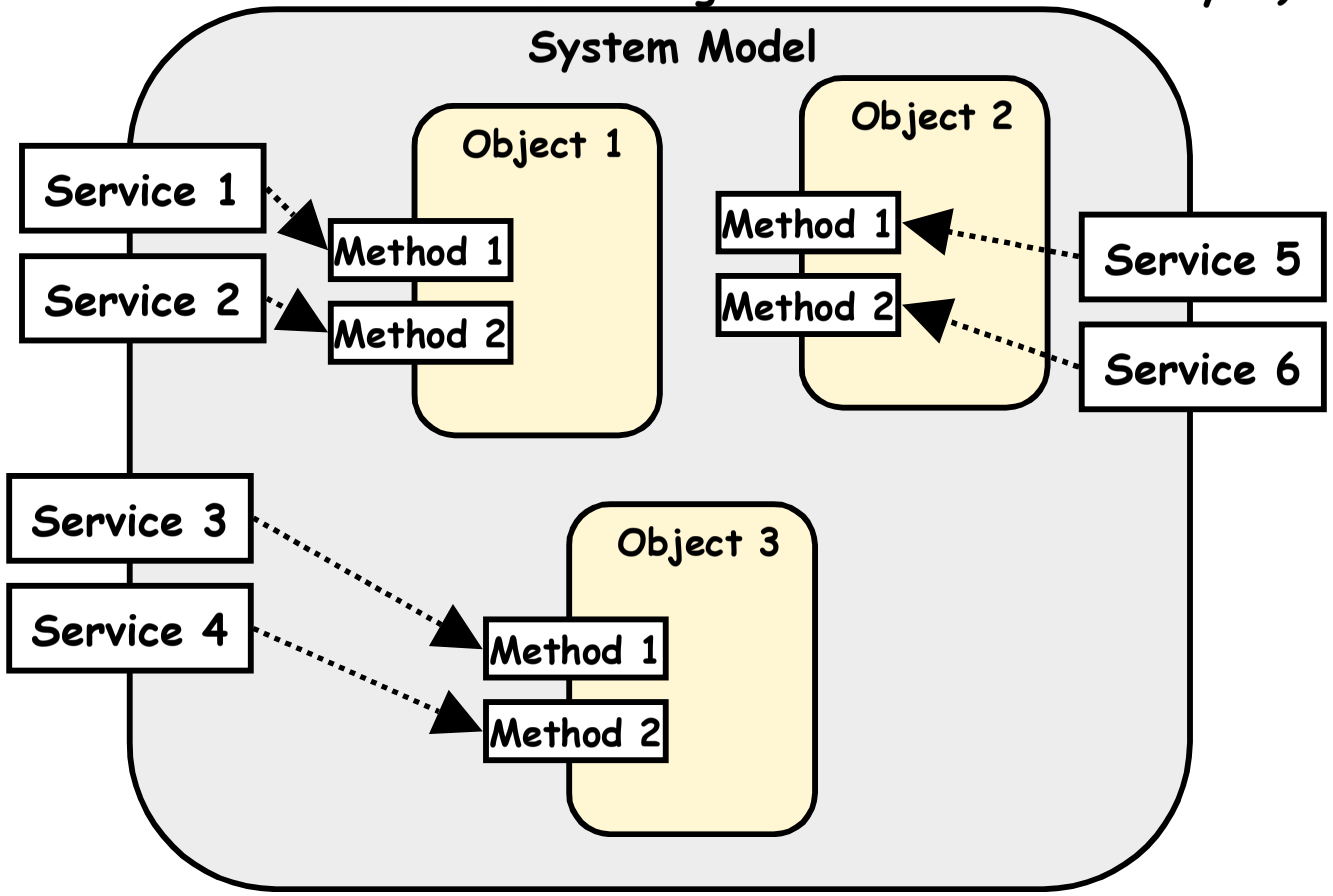
## Aggregation

Can describe relationships between parts and the whole

# Information Hiding

## Objects can contain other objects

(compare with hierarchies of dataflow diagram in Structured Analysis)



# Nearly anything can be an object…

*See also:* van Vliet 1999, section 12.3

## External Entities

…that interact with the system being modeled

E.g. people, devices, other systems

## Things

…that are part of the domain being modeled

E.g. reports, displays, signals, etc.

## Occurrences or Events

…that occur in the context of the system

E.g. transfer of resources, a control action, etc.

## Roles

played by people who interact with the system

## Organizational Units

that are relevant to the application

E.g. division, group, team, etc.

## Places

…that establish the context of the problem being modeled

E.g. manufacturing floor, loading dock, etc.

## Structures

that define a class or assembly of objects

E.g. sensors, four-wheeled vehicles, computers, etc.

## *Some things cannot be objects:*

*procedures (e.g. print, invert, etc)*

*atomic attributes (e.g. blue, 50Mb, etc)*

# Selecting Objects

***Source:*** *Adapted from Pressman, 1994, p244*

## Need to choose which candidate objects to include in the analysis

Coad & Yourdon suggest each object should satisfy (most of) the following criteria:

- **Retained information**: Does the system need to remember information about this object?
- **Needed Services**: Does the object have identifiable operations that change the values of its attributes?
- **Multiple Attributes**: If the object only has one attribute, it may be better represented as an attribute of another object
- **Common Attributes**: Does the object have attributes that are shared with all occurrences of the object?
- **Common Operations**: Does the object have operations that are shared with all occurrences of the object?

**Note:** External entities that produce or consume information essential to the system are nearly always objects

Many candidate objects will be eliminated or combined

# Variants

*See also:* van Vliet 1999, section 12.3

## Coad-Yourdon

- Developed in the late 80's
- Five-step analysis method

## Shlaer-Mellor

- Developed in the late 80's
- Emphasizes modeling information and state, rather than object interfaces

## Fusion

- Second generation OO method
- Introduced message sequence charts

## Unified Modeling Language (UML)

- Third generation OO method
- An attempt to combine advantages of previous methods

*Source: Adapted from Pressman, 1994, p242 and Davis 1990, p98-99*

# Five Step Process: Coad-Yourdon

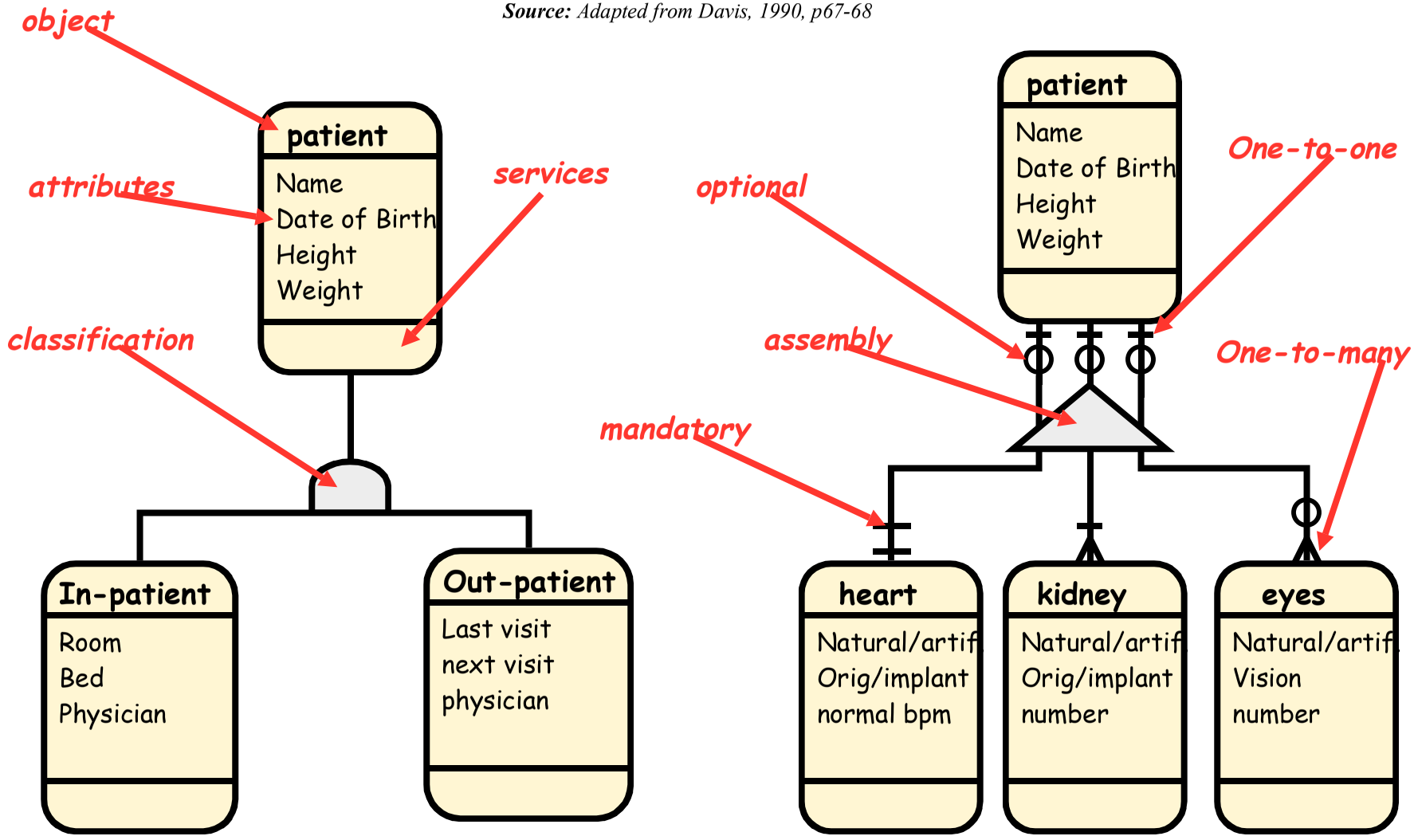
1. Identify Objects & Classes *(i.e. 'is_a' relationships)*
2. Identify Structures *(i.e. 'part_of' relationships)*
3. Define Subjects
   - A more abstract view of a large collection of objects
   - Each classification and assembly structure become one subject
   - Each remaining singleton object becomes a subject *(although if there a many of these, look for more structure!)*
   - Subject Diagram shows only the subjects and their interactions
4. Define Attributes and instance connections

5a. Define services - 3 types:
   - Occur (create, connect, access, release) *These are omitted from the model as every object has them*
   - Calculate (when a calculated result from one object is needed by another)
   - Monitor (when an object monitors for a condition or event)

5b. Define message connections
   - These show how services of one object are used by another
   - Shown as dotted lines on object and subject diagrams
   - Each message may contain parameters

# Coad Object diagrams

*Source: Adapted from Davis, 1990, p67-68*

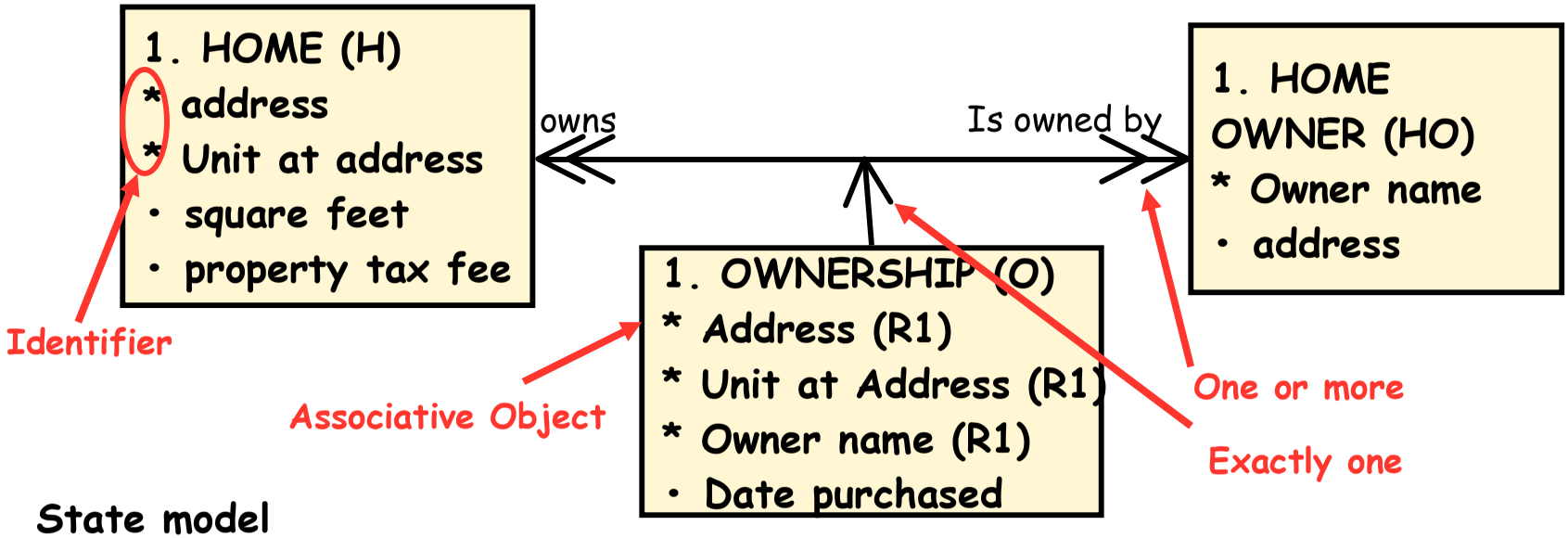


# Shlaer-Mellor

## Three analysis models:

### Information Model

models objects, relationships, and attributes of objects and relationships
uses *associative objects* to represent relationships between other objects.
E.g. 'title' is an object that represents the relationship between 'owner' and 'car'




### State model

Uses StateCharts to show the lifecycle of each object
Each object may be continuous or born-and-die (object is created & destroyed)

### Process model

representation of each service *('action')* of an object
Uses standard Dataflow Diagrams to show information used

# Fusion

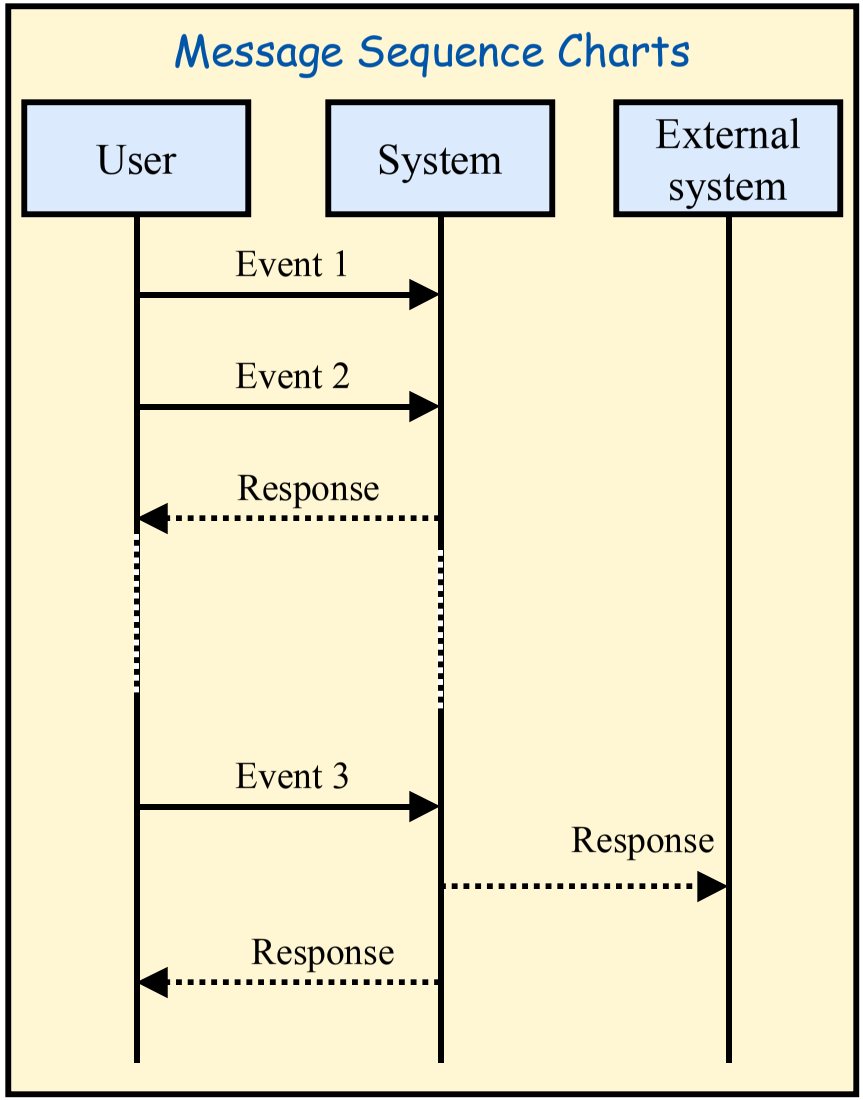
## Combines several OO methods

## Analysis phase:

- Object model
  - like Shlaer-Mellor
- Operation model
  - formal definition of each operation, including pre- and post- conditions
- Lifecycle model
  - specifies admissible sequences of interactions between system & environment
- Interaction model
  - = operation model + lifecycle model

## Message Sequence Charts

help to develop the interaction model



# Unified Modeling Language (UML)

## Third generation OO method

- Booch, Rumbaugh & Jacobson are principal authors
  - Still in development
  - Attempt to standardize the proliferation of OO variants
- Is purely a notation
  - No modeling method associated with it!
- But has been accepted as a standard for OO modeling
  - But is primarily owned by Rational Corp. (who sell lots of UML tools and services)

## Has a standardized meta-model

- Class diagrams
- Use case diagrams
- Message trace diagrams
- Object message diagrams
- State Diagrams (uses Harel's statecharts)
- Module Diagrams
- Platform diagrams

# Evaluation of OOA

## Advantages of OO analysis for RE

Fits well with the use of OO for design and implementation

- Transition from OOA to OOD 'smoother' than from SA to SD (but is it?)

Removes emphasis on functions as a way of structuring the analysis

Avoids the fragmentary nature of structured analysis

- object-orientation is a coherent way of understanding the world

## Disadvantages

Emphasis on objects brings an emphasis on static modeling

- although later variants have introduced dynamic models

Not clear that the modeling primitives are appropriate

- are objects, services and relationships really the things we need to model in RE?

Strong temptation to do design rather than problem analysis

Too much marketing hype

- and false claims - e.g. no evidence that objects are a more natural way to think

# References

van Vliet, H. "Software Engineering: Principles and Practice (2nd Edition)" Wiley, 1999.

chapter 12 is a thorough overview of object oriented analysis and design. van Vliet introduces all the main notations of UML, and describes several older methods too.

Svoboda, C. P. "Structured Analysis". In Thayer, R. H and Dorfman, M. (eds.) "Software Requirements Engineering, Second Edition". IEEE Computer Society Press, 1997, p255-274

Excellent overview of the history of structured analysis, and a comparison of the variants

Davis, A. M. "Software Requirements: Analysis and Specification". Prentice-Hall, 1990.

This is probably the best textbook around on requirements analysis, although is a little dated now.