Lecture 6: Requirements Modeling II

Last Week:
Modeling and Analysis (I)
General Modeling Issues
Modeling Goals, Organizations, and Non-Functional Requirements

This Week:
Modeling and Analysis (II)
Modeling Functionality
Structured Analysis
Object Oriented Analysis

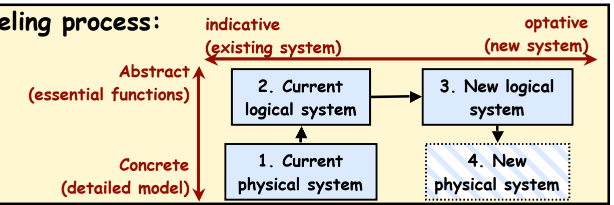
Next Week:
Modeling and Analysis (III)
Formal Modeling Techniques
Formal Reasoning

Structured Analysis

→ Definition

- Structured Analysis is a data-oriented approach to conceptual modeling
- Common feature is the centrality of the dataflow diagram
- Mainly used for information systems
  - variants have been adapted for real-time systems

→ Modeling process:



- Model of current physical system only useful as basis for the logical model
- Distinction between indicative and optative models is very important:
  - Must understand which requirements are needed to continue current functionality, and which are new with the updated system

Central Concepts

Source: *Adapted from Svoboda, 1990, p257*

→ Process (data transformation)

- activities that transform data
- related by dataflows to other processes, data store, and external entities.

→ Data flow

- indicate passage of data from output of one entitie to input of another
- represent a data group or data element

→ Data store

- a place where data is held for later use
- Data stores are passive: no transformations are performed on the data

→ External entity

- An activity outside the target system
- Acts as source or destination for dataflows that cross the system boundary
- External entities cannot interact directly with data stores

→ Data group

- A cluster of data represented as a single dataflow
- Consists of lower level data groups, or individual elements

→ Data element

- a basic unit of data

Modeling tools

Source: *Adapted from Svoboda, 1990, p258-263*

→ Data flow diagram

- Context diagram ("Level 0")
  - whole system as a single process
- intermediate level DFDs decompose each process
- functional primitives are processes that cannot be decomposed further

→ Data dictionary

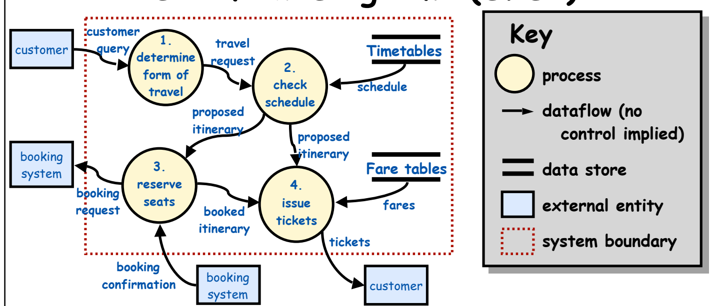
- Defines each data element and data group
- Use of BNF to define structure of data groups

→ Primitive Process Specification

- Each functional primitive has a "mini-spec"
- these define its essential procedural steps
- Expressed in English narrative, or some form of pseudo-code

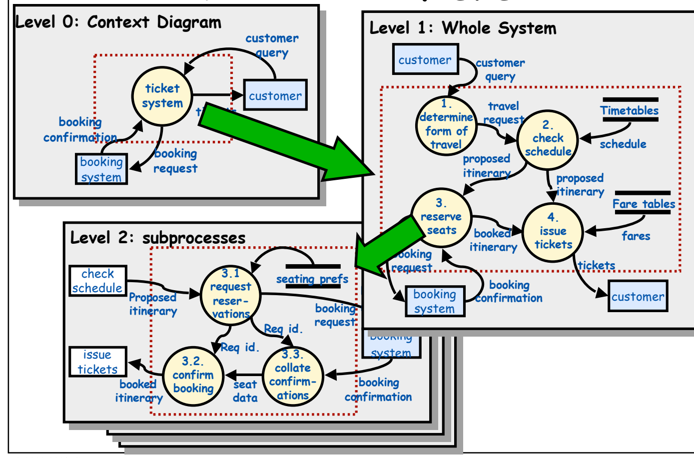
→ Structured Walkthrough

Dataflow Diagrams (DFDs)



→ **Notes:**
- every process, flow, and datastore must be labeled
- representation is hierarchical
  - each process will be represented separately as a lower level DFD
- processes are normally numbered for cross reference
- processes transform data
  - can't have the same data flowing out of a process as flows into it

Hierarchies of DFDs



Data Dictionary & Process Specs

***Source:** Adapted from Svoboda, 1990, p262-4*

Example Data Dictionary

Mailing Label =
  customer_name +
  customer_address

customer_name =
  customer_last_name +
  customer_first_name +
  customer_middle_initial

customer_address =
  local_address +
  community_address + zip_code

local_address =
  house_number + street_name +
  (apt_number)

community address =
  city_name + [state_name |
  province_name]

Example Mini-Spec

FOR EACH Shipped-order-detail
  GET customer-name + customer-address
  FOR EACH part-shipped
    GET retail-price
    MULTIPLY retail-price by quantity-shipped
    TO OBTAIN total-this-order
CALCULATE shipping-and-handling
ADD shipping-and-handling TO total-this-order
  TO OBTAIN total-this-invoice
PRINT invoice

DFD variants

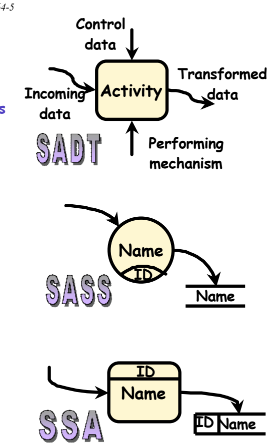
***Source:** Adapted from Svoboda, 1990, p264-5*

→ **Structured Analysis and Design Technique (SADT)**
- Developed by Doug Ross in the mid-70's
- Uses activity diagrams rather than dataflow diagrams
- Distinguishes control data from processing data

→ **Structured Analysis and System Specification (SASS)**
- Developed by Yourdon and DeMarco in the mid-70's
- 'classic' structured analysis

→ **Structured System Analysis (SSA)**
- Developed by Gane and Sarson
- Notation similar to Yourdon & DeMarco
- Adds data access diagrams to describe contents of data stores

→ **Structured Requirements Definition (SRD)**
- Developed by Ken Orr in the mid-70's
- Introduces the idea of building separate models for each perspective and then merging them



SASS methodology

Source: *Adapted from Davis, 1990, p83-86*

1. **Study current environment**
   - draw DFD to show how data flows through current organization
   - label bubbles with names of organizational units or individuals
2. **Derive logical equivalents**
   - replace names (of people, roles,…) with action verbs
   - merge bubbles that show the same logical function
   - delete bubbles that don't transform data
3. **Model new logical system**
   - Modify logical DFD to show how info will flow once new system is in place
     - …but don't distinguish (yet) which components will be automated
4. **Define a number of automation alternatives**
   - document each as a physical DFD
   - Analyze each with cost/benefit trade-off
   - Select one for implementation
   - Write the specification

Alternative Process Model: SRD

Source: *Adapted from Davis, 1990, p72-75*

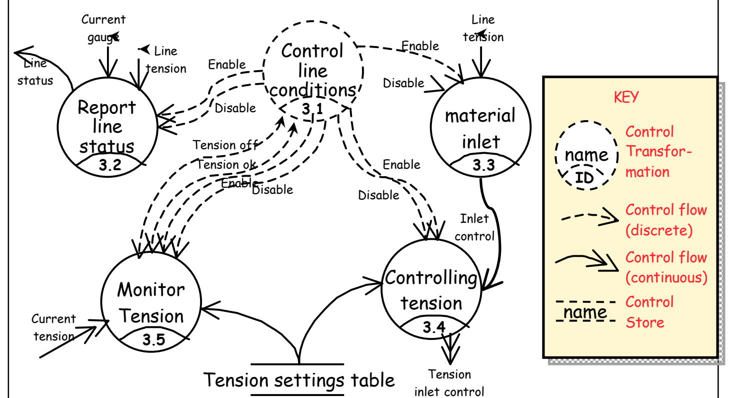
1. **Define a user-level DFD**
   - interview each relevant individual in the current organization
     - actually a role, rather than an individual
   - Identify the inputs and outputs for that individual
   - Draw an 'entity diagram' showing these inputs and outputs
2. **Define a combined user-level DFD**
   - Merge all alike bubbles to create a single diagram
   - Resolve inconsistencies between perspective
3. **Define the application-level DFD**
   - Draw the system boundary on the combined user-level DFD
   - Then collapse everything within the boundary into a single process
4. **Define the application-level functions**
   - label inputs and outputs to show the order of processing for each function
     - I.e. for function A, label the flows that take part in A as A1, A2, A3,...

Later developments

- **Later work recognized that:**
  - development of both current physical and current logical models is overkill
  - top down development doesn't always work well for complex systems
  - entity-relationship diagrams are useful for capturing complex data
- **Structured Analysis / Real Time (SA/RT)**
  - Developed by Ward and Mellor in the mid-80's
  - Extends structured analysis for real-time systems
    - Adds control flow, state diagrams, and entity-relationship models
- **Modern Structured Analysis**
  - Captured by Yourdon in his 1989 book
  - Uses two models: the environmental model and the behavioral model
    - together these comprise the *essential* model
  - Includes plenty of advice culled from many years experience with structured analysis

Real-time extensions

Source: *Adapted from Svoboda, 1990, p269*



Evaluation of SA techniques

Source: *Adapted from Davis, 1990, p174*

→ Advantages
- Facilitates communication.
- Notations are easy to learn, and don't require software expertise
- Clear definition of system boundary
- Use of abstraction and partitioning
- Automated tool support
  - e.g. CASE tools provide automated consistency checking

→ Disadvantages
- Little use of projection
  - even SRD's 'perspectives' are not really projection
- Confusion between modeling the problem and modeling the solution
  - most of these techniques arose as design techniques
- These approaches model the system, *but not its application domain*
- Timing issues are completely invisible

Object Oriented Analysis

→ Background
- Model the requirements in terms of objects and the services they provide
- Grew out of object oriented design
  - OOD partitions a program in a different way from structured programming
  - Result was a poor fit moving from Structured Analysis to Object Oriented Design

→ Motivation
- OO 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
  - Hence a structured analysis model will get out of date, but an object oriented model will not…
  - …hence the claim that object-oriented designs are more maintainable
- OO 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 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

→ Methods (or '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

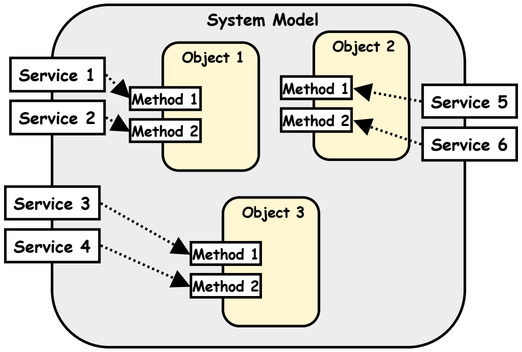
→ Classification (using inheritance)
- Classes capture commonalities of a number of objects
  - Each subclass inherits attributes and methods from its parent
  - Forms an 'is_a' hierarchy
- Child class may 'specialize' the parent class
  - by adding additional attributes & methods
  - by replacing an 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…

Source: Adapted from Pressman, 1994, p242

- 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)*
- *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 during modeling

Variants

- 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 use-cases
- Unified Modeling Language (UML)
  - Third generation OO method
  - An attempt to combine advantages of previous methods

Coad-Yourdon

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

→ Five Step Process:

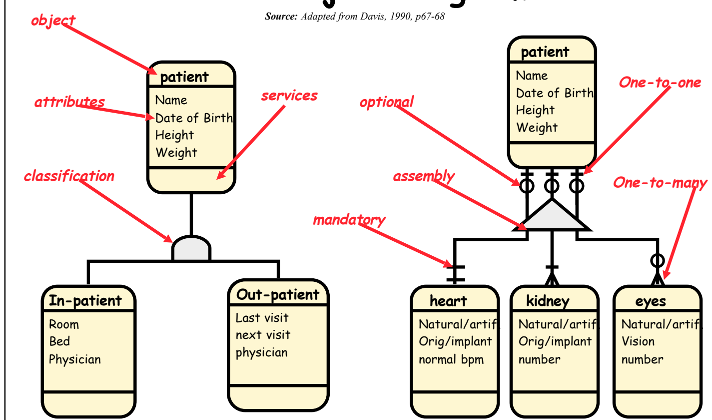
1. Identify Objects & Classes *(i.e. 'is_a' relationships)*
2. Identify Structures *(i.e. 'part_of' relationships)*
3. Define Subjects
    - ➢ A more abstract view of a large collection of objects
    - ➢ Each classification and assembly structure become one subject
    - ➢ Each remaining singleton object 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

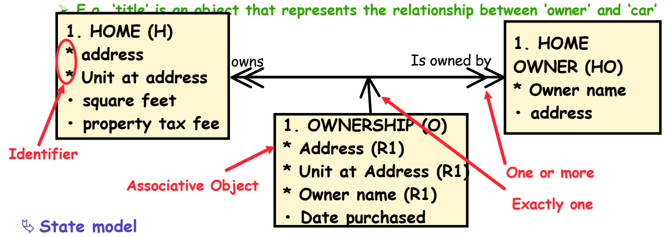
Source: *Adapted from Davis, 1990, p67-68*



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

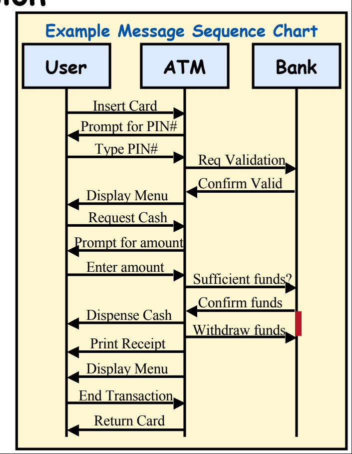
→ Combines several OO methods

→ 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

- ✎ Used in the interaction model

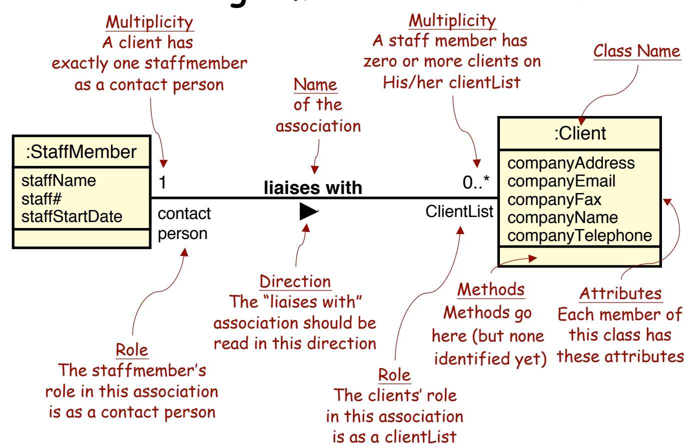


Unified Modeling Language

→ **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**
- Use case diagrams *(see lecture 3)*
- Class diagrams
- Message sequence charts
- Activity diagrams
- State Diagrams (uses Harel's statecharts)
- Module Diagrams
- Platform diagrams

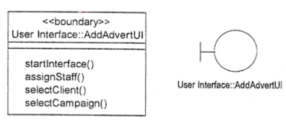
Class diagrams and associations



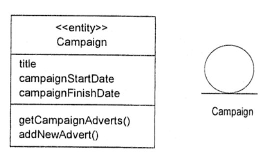
Class Stereotypes

Source: *Examples from Bennett, McRobb & Farmer, 2002*

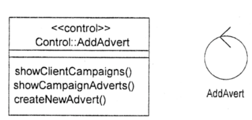
→ **Boundary Classes**
- Model the interactions between the system and its actors
  - Mainly used for interface design



→ **Entity Classes**
- The information represented by the system
  - Objects in the application domain that the system needs to know about



→ **Control Classes**
- Represent coordination, sequencing, transactions, & control of other objects
  - E.g. one for each use case



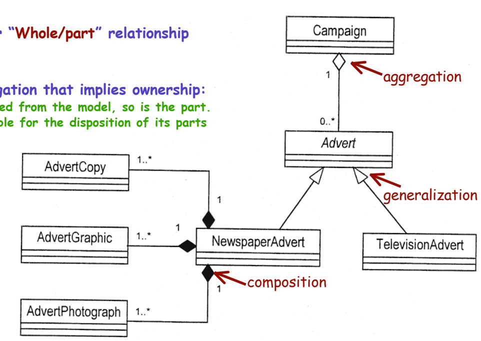
Generalization and Aggregation

Source: *Examples from Bennett, McRobb & Farmer, 2002*

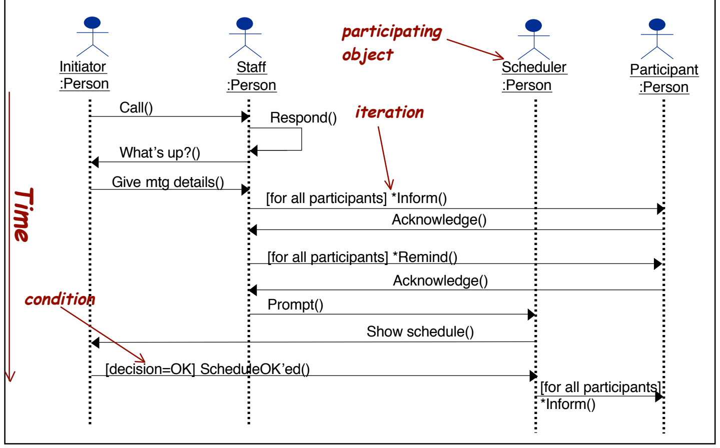
→ **Generalization**
- Subclasses inherit attributes, associations, & operations from the superclass
- A subclass may override an inherited aspect

→ **Aggregation**
- This is the "**Has-a**" or "**Whole/part**" relationship

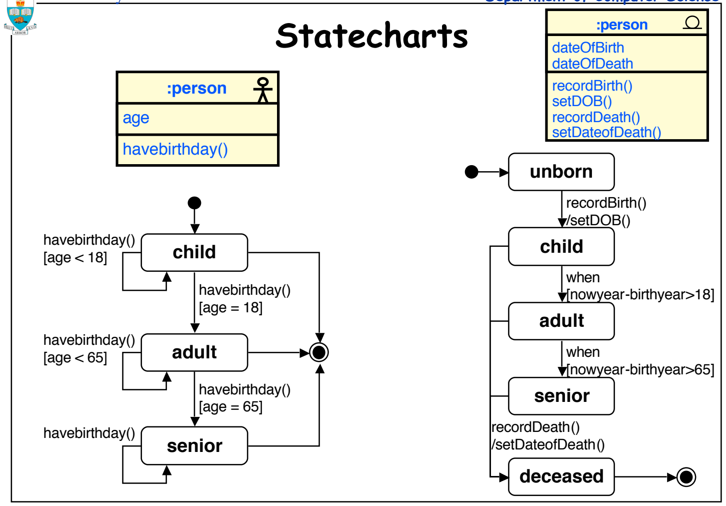
→ **Composition**
- Strong form of aggregation that implies ownership:
  - if the whole is removed from the model, so is the part.
  - the whole is responsible for the disposition of its parts



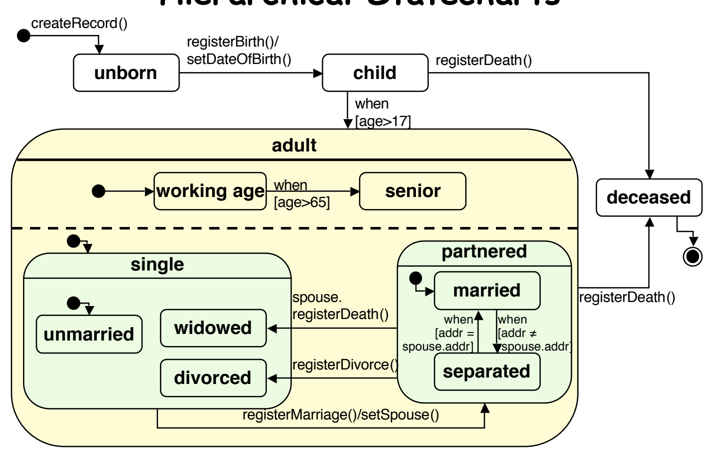
Example Sequence Diagram



Statecharts



Hierarchical Statecharts



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
- Fragmentation of the analysis
  - E.g. reliance on use-cases means there is no "big picture" of the user's needs
- Too much marketing hype!
  - and false claims - e.g. no evidence that objects are a more natural way to think