Lecture 6: Requirements Modeling II

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:

- Abstract (essential functions)
- Concret (detailed model)

1. Current physical system
2. Current logical system
3. New logical system
4. New physical system

- 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

- 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 entity 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

- 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)

Key
- process
- dataflow (no control implied)
- data store
- external entity
- system boundary

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

Data Dictionary & Process Specs

Example Data Dictionary
Mailing Label = customer_name + customer_address
customer_name = customer_last_name + customer_first_name
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
- 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

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

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
Evaluation of SA techniques

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.

Source: Adapted from Davis, 1990, p174

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 through ‘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.

University of Toronto
Department of Computer Science
© 2000-2003, Steve Easterbrook
Information Hiding

- Objects can contain other objects
  (compare with hierarchies of dataflow diagram in Structured Analysis)

Service 1
Method 1
Method 2
Object 1
Method 1
Method 2
Service 5
Method 1
Method 2
Service 6

Service 2
Method 1
Method 2
Object 2
Method 1
Method 2

Service 3
Method 1
Method 2
Object 3
Method 1
Method 2
Service 4

Nearly anything can be an object...

- 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, $\text{RGB}(0,0,255)$, etc)

Selecting Objects

- 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
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 a subject
   - Each remaining singleton object becomes a subject (although if there are 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

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 State Charts 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
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

- StaffMember
  - staffName
  - staffNumber
  - staffStartDate

- Client
  - companyAddress
  - companyEmail
  - companyFax
  - companyName
  - companyTelephone

- liaises with
  - contact person
  - ClientList

- Multiplicity
  - A client has exactly one staff member as a contact person

- Role
  - The staff member's role in this association is as a contact person
  - The clients' role in this association is as a clientList

- Direction
  - The "liaises with" association should be read in this direction

Attributes

- Each member of this class has these attributes

Class Stereotypes

- 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

- 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

Source:
- Examples from Bennett, McRobb & Farmer, 2002
Example Sequence Diagram

Call()
respond()

What’s up?

(give mtg details)

Respond()

iteration

for all participants

Inform()

[for all participants]

Remind()

Prompt()

Show schedule()

[decision=OK]

Schedule OK’ed

Initiator

Participant

Staff

Scheduler

Condition

Iteration

Time

Statecharts

havebirthday()

[age = 18]

child

havebirthday()

[age < 18]

unborn

recordBirth() / setDOB()

havebirthday()

[age < 65]

adult

havebirthday()

[age = 65]

recordDeath() / setDateofDeath()

havebirthday()

[age > 65]

senior

havebirthday()

[age > 18]

recordMarriage() / setSpouse()

Hierarchical Statecharts

createRecord()

registerBirth() / setDOB()

registerDeath()

unborn

child

working age

when

age > 17

adult

when

age > 65

working age

single

when

age > 65

working age

partnered

married

when

age > 65

single

unmarried

widowed

divorced

spouse.

registerDeath() / setDOB()

registerMarriage() / setSpouse()

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