

ECE 452- Introduction to Software Engineering

Lecture 10: Object-Oriented Analysis (Conceptual Model)

Manish Parashar
parashar@ece.rutgers.edu
Department of Electrical & Computer Engineering
Rutgers University

The Unified Analysis Process & UML

- ◆ Use Case - What are the domain processes ?
 - Use Case Diagram
 - ◆ high-level/expanded, essential/real
- ◆ Conceptual Model - What are the domain concepts, terms ?
 - Class Diagram (conceptual)
 - ◆ classes, associations, attributes
- ◆ System Sequence Diagram - What are the system events and operations ?
 - Interaction Diagram - Sequence Diagram
- ◆ Contracts - What do the system operations do ?
 - Contract Specs.

ECE 452 - Introduction to Software Engineering

Introduction

- ◆ Conceptual Model
 - illustrates meaningful concepts
 - most important artifact in OO analysis
 - need use cases as input
 - ◆ but can develop uses cases as you do conceptual model
- ◆ Question:
 - Provide names for the uses cases you found
 - ◆ let's share this information now (use whiteboard)

ECE 452 - Introduction to Software Engineering

Domain Analysis

- ◆ Object oriented analysis at the “business area level”
- ◆ Objective: Create a library of reusable components for a particular domain
- ◆ Ongoing activity of the software process (not tied to a specific project)
- ◆ Domain analysis model specifies objects and classes that characterize the domain

ECE 452 - Introduction to Software Engineering

Domain Analysis

- ◆ Define domain to be investigated
- ◆ Categorize the items extracted from the domain
- ◆ Collect a representative sample of applications in the domain
- ◆ Analyze each application in the sample
- ◆ Develop an analysis model for the objects
- ◆ Define reuse guidelines

ECE 452 - Introduction to Software Engineering

Conceptual Models

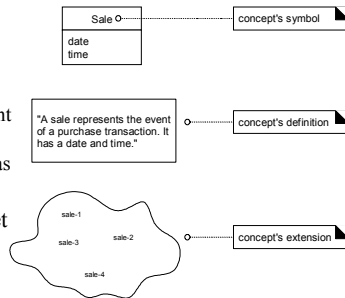
- ◆ Conceptual Model
 - created for the use cases of the current cycle in the language of the problem domain
- ◆ Must identify
 - concepts - objects in our system
 - associations between concepts
 - ◆ is part of, contains, manages, is-a, ...
 - attributes of concepts
 - ◆ instance variables needed to hold state

ECE 452 - Introduction to Software Engineering

Example Concept

◆ POST Sale

- symbol: Sale
- definition
represents the event of a purchase transaction, and has a date and time
- extension: the set of all sales

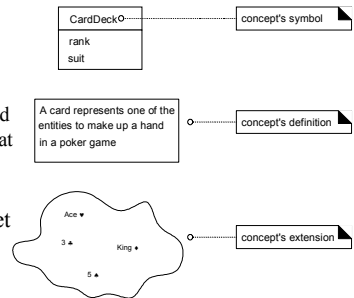


ECE 452 - Introduction to Software Engineering

Example Concept

◆ POKER Card

- symbol: Card
- definition:
represents one card in a poker deck that has a rank (2..14), and a suit ♣♦♥♠
- extension: the set of all cards



ECE 452 - Introduction to Software Engineering

Conceptual model and partitioning

- ◆ Structured Analysis (still in use today)
 - Divide and Conquer
 - at the function level
- ◆ Object-Oriented Analysis (still growing)
 - Partition
 - at the level of concepts (objects)

ECE 452 - Introduction to Software Engineering

Identifying Concepts

- ◆ One strategy for getting concepts
 - Look for the concepts in the problem specification and in your uses cases
 - read the final project specification
 - It is better to have too many at first than too few
 - specifying too many concepts is good
 - is is better too have too many than too few
 - concepts may have no attributes
- ◆ Try not to think of Java classes as you do this
 - later on, think of some classes to implement

ECE 452 - Introduction to Software Engineering

Concept Category

- ◆ Physical or tangible things
 - **POST** A Point of Sale Terminal
- ◆ Specifications
 - **POST** ProductDescription
- ◆ Transactions
 - **POST** Sale, Payment

ECE 452 - Introduction to Software Engineering

More Concept Categories

- ◆ Things in a container
 - **POST** Sales Line Item
- ◆ Abstract noun concepts
 - **POST** Waiting line rage
- ◆ Events
 - **POST** Sale

ECE 452 - Introduction to Software Engineering

More Concept Categories

- ◆ Rules and Policies
 - POST RefundPolicy
- ◆ Organizations
 - POST SalesDepartment

For more, see pages 91-93

ECE 452 - Introduction to Software Engineering

Concepts? page 96-97

- ◆ Use vocabulary of the users
 - Users are not usually present for college programming projects
 - However, Poker has users
 - ◆ the users are you *this isn't a Library with Patrons and Librarians*
- ◆ Do not include irrelevant things
 - no casino, no pit boss
- ◆ Do not include things that don't belong
 - no Lottery System, no Starship Enterprise, no pit bull

ECE 452 - Introduction to Software Engineering

Finding Concepts

- ◆ Make a list of candidate concepts (objects)
 - look for noun phrases
 - ◆ be careful to avoid ambiguities
 - ◆ see page 95 POST Item, Store, Sale, Payment, Customer

ECE 452 - Introduction to Software Engineering

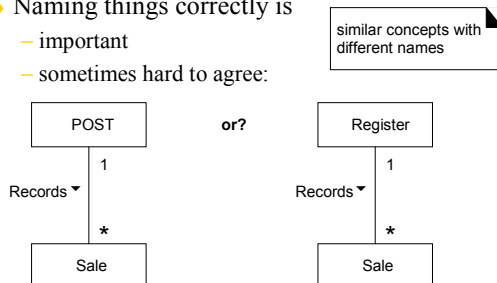
Common Mistake

- ◆ Do not mistake an attribute for a concept
 - Attributes are numbers, booleans, strings, date
- ◆ All other things are likely a concept
 - if in doubt, make it a concept
- ◆ Example: Payment in the POST problem could be thought of as the amount of a Sale
 - a primitive attribute like 9.75
 - But a payment could be made by check, cash, credit card, could be in various currencies of the world, could require change
 - This outgrows being a simple attribute

ECE 452 - Introduction to Software Engineering

Resolving Similar Concepts

- ◆ Naming things correctly is
 - important
 - sometimes hard to agree:

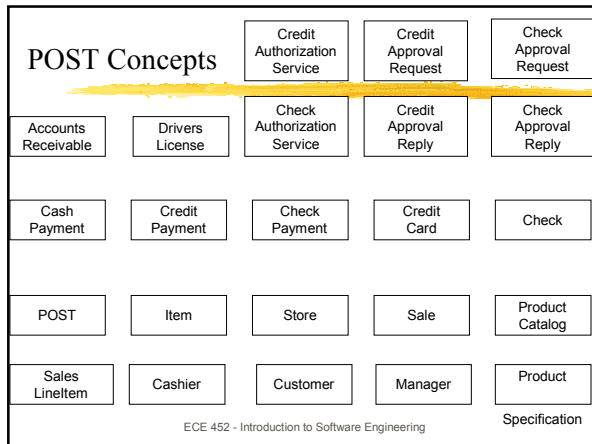


ECE 452 - Introduction to Software Engineering

Abstract Concepts

- ◆ Some concepts don't feel real *and that is okay*
 - Telecommunications
 - ◆ Connection, Route, Protocol
 - Poker
 - ◆ Strategy, View
 - Encoding images with some compression scheme or encrypting messages for privacy
 - ◆ Compression, Encryption, Privacy
- ◆ Concepts in the problem domain exist even if they do not seem to be part of the real world

ECE 452 - Introduction to Software Engineering



Defining Terms in the UML

- ◆ The Software class diagram will look a lot like the conceptual model
 - ◆ You will see associations, boxes, multiplicity...
 - ◆ Larman uses concept to refer to real world things *and unreal things*
 - ◆ Classes refer to software specifications and implementations. UML defines class as
 - a description of a set of objects that share the same attributes, relationships, and semantics
- the three Amigos: Grady Booch, Ivar Jacobsen, Jim Rumbaugh*

ECE 452 - Introduction to Software Engineering

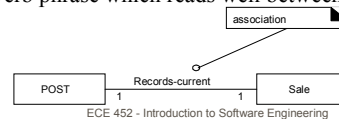
Other UML terms

- ◆ Operation
 - a service that can be requested from an object to affect behavior
- ◆ Type
 - either a class or an interface *specification with no methods*
- ◆ Interface
 - the set of externally visible operations
 - Could mean a Java interface

ECE 452 - Introduction to Software Engineering

Adding Associations

- ◆ Next step in the conceptual model
 - adding associations between two concepts
 - an association is
 - a relationship between concepts that indicates some meaningful and interesting connection
 - Name the association with a hyphen connected verb phrase which reads well between concepts



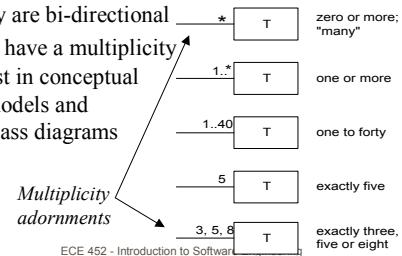
Associations

- ◆ Associations imply
 - our knowledge of the relationship must be preserved for some time (1 ms to forever)
 - between what objects do we need to remember a relationship?
 - POST: does a driver's license remember a product catalog?
 - Poker: does a game need to remember it's Player(s)?
 - Poker: does the winner need to know the card image collection?

ECE 452 - Introduction to Software Engineering

The UML Association Notation

- ◆ UML Association:
 - a line between two concepts and a name
 - they are bi-directional
 - can have a multiplicity
 - exist in conceptual models and class diagrams



Possible Associations

- ◆ A is a physical part of B *Drawer - POST*
- ◆ A is a logical part of B *SalesLineItem - Sale*
- ◆ A is physically contained in B *POST-Store*
- ◆ A is logically contained in B *ItemDescription-Catalog*
- ◆ A is recorded in B *Sale - POST*
- ◆ A uses or manages B *Cashier - POST*
- ◆ A communicates with B *Customer - Cashier*
- ◆ A is related to a transaction B *Customer - Payment*
- Draw some poker associations now
 - ◆ Let's try to get one for each of the bullets above

ECE 452 - Introduction to Software Engineering

Associations

- ◆ Some useful associations
 - A is a physical or logical part of B
 - A is physically or logically contained in/on B
 - A is recorded in B

ECE 452 - Introduction to Software Engineering

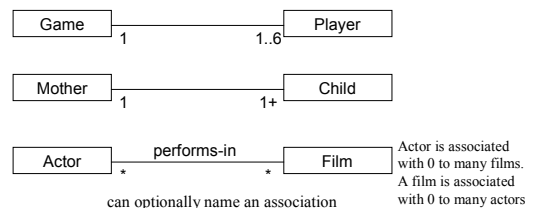
Association Guidelines

- ◆ Concepts are more important than associations
- ◆ Focus on *need to know* associations
 - the relationship must be preserved for some time
- ◆ Better to have too few associations than too many

ECE 452 - Introduction to Software Engineering

Multiplicity

- ◆ Multiplicity defines how many instances of type A can be associated with one instance of type B at some point *can differ*



ECE 452 - Introduction to Software Engineering

Depends on Context

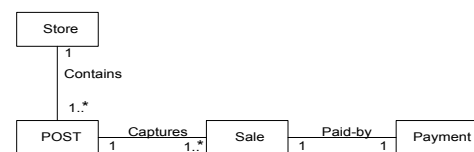
- ◆ Are all three associations possible?



ECE 452 - Introduction to Software Engineering

Association Names Upcase / hyphenate

- ◆ Try reading this *Type-VerbPhrase-Type*



- ◆ Not yet worrying about messages, instance variables, data flow, or software objects
- ◆ Just show some concepts are connected

ECE 452 - Introduction to Software Engineering

No Implementation Yet

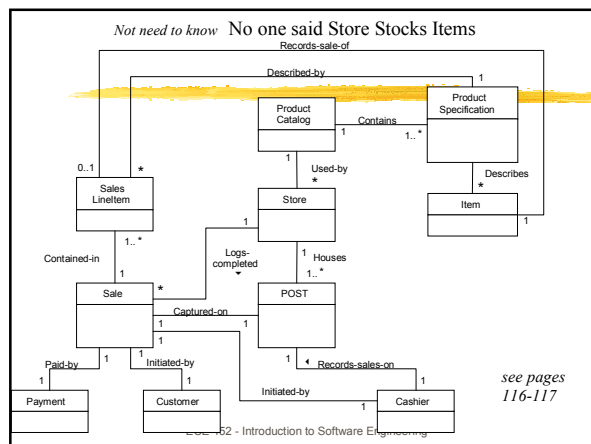
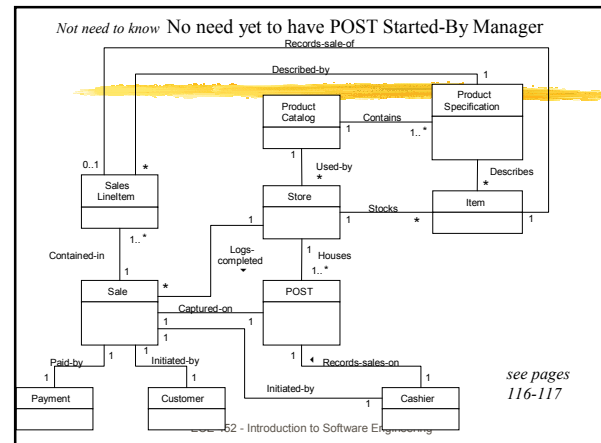
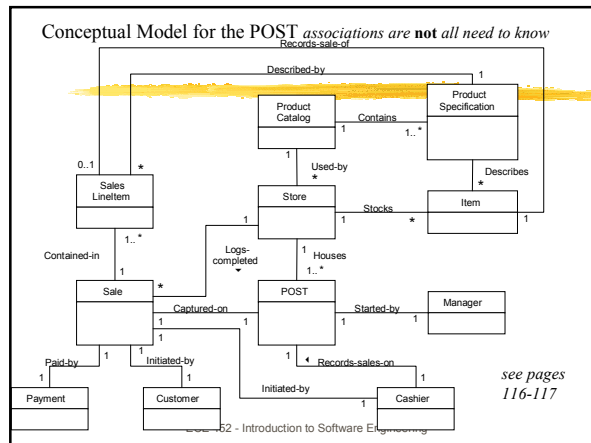
- ◆ The associations may later become implemented in software in terms of visibility
 - Related as an instance variable of a class
 - Related as an argument in a message
 - Related through an inheritance relationship
- ◆ Some associations will not be implemented
 - Others will be discovered during coding
- ◆ Conceptual Model should have associations suggested by requirements and use cases

ECE 452 - Introduction to Software Engineering

Any unforgettable Relationships?

- ◆ POST
 - Post Captures Sale
 - Sale Paid-By Payment
 - ProductCatalog Records Specification

ECE 452 - Introduction to Software Engineering



Need to Know vs. Comprehension

- ◆ Could do a strict need-to-know model
 - bounded by current requirements only
 - but it may not convey a full understanding
- ◆ By eliminating concepts and associations that are not currently required need to know
 - the conceptual model may not communicate key ideas and relationships
- ◆ Make your conceptual model so it
 - has need to know requirements *what the system must do*
 - clearly communicates important concepts
 - ◆ could have View as a concept

ECE 452 - Introduction to Software Engineering

Summary

- ◆ Doubting whether to keep a concept?
 - keep it
- ◆ Doubting whether to keep an association?
 - drop it
- ◆ Do not keep derived associations
 - Where A relates to B and B relates to C usually leave out the A to C relationship
- ◆ Does model satisfy all need to know associations and still present a clear complete picture?

ECE 452 - Introduction to Software Engineering

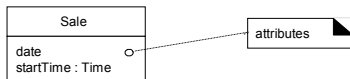
Identifying Attributes

- ◆ The final piece to the conceptual model is to add the attributes needed to support use cases
- ◆ *Attribute*: a logical data value of an object
 - not necessarily in the final design, though some attributes will end up in the code
- ◆ Example: A bank customer has an account ID and a balance
 - both could be attributes in the conceptual model
 - both could be instance variables in a class diagram

ECE 452 - Introduction to Software Engineering

Keep attributes simple

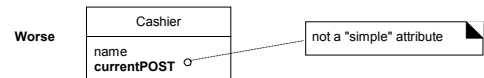
- ◆ Look for simple data types as attributes
 - boolean, date, number, string *like data base primitives*
- ◆ Do not use complex ideas as attributes
- ◆ UML puts attributes in 2nd row of concept box



ECE 452 - Introduction to Software Engineering

Attribute or Association?

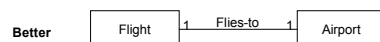
- ◆ Easy to mistake a concept as an attribute



ECE 452 - Introduction to Software Engineering

Attribute or Association?

- ◆ Attributes should be simple, if complex, write it as a concept with an association



ECE 452 - Introduction to Software Engineering