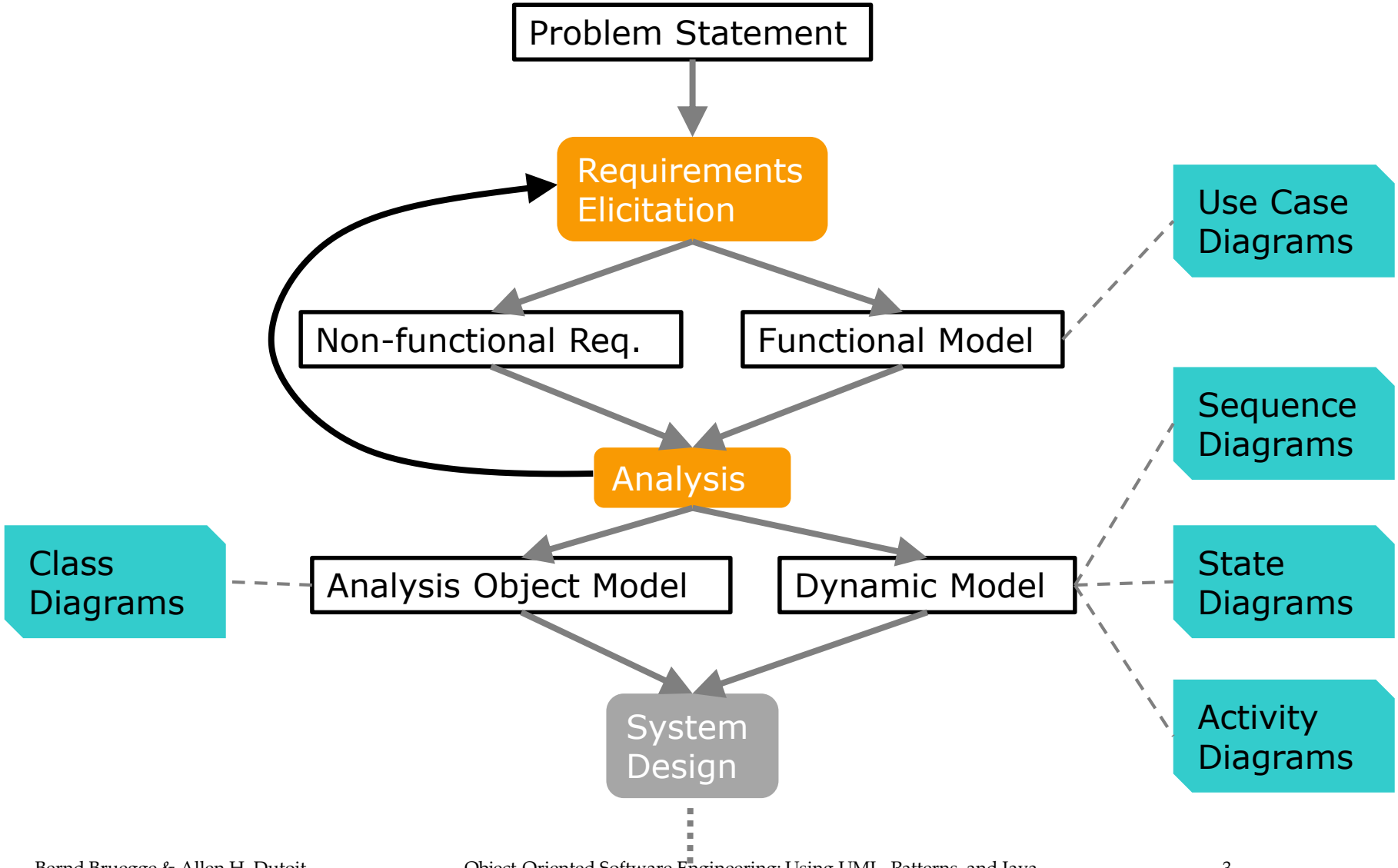
Object-Oriented Software Engineering
Using UML, Patterns, and Java

Chapter 5: Analysis, Object Modeling

Ambiguity

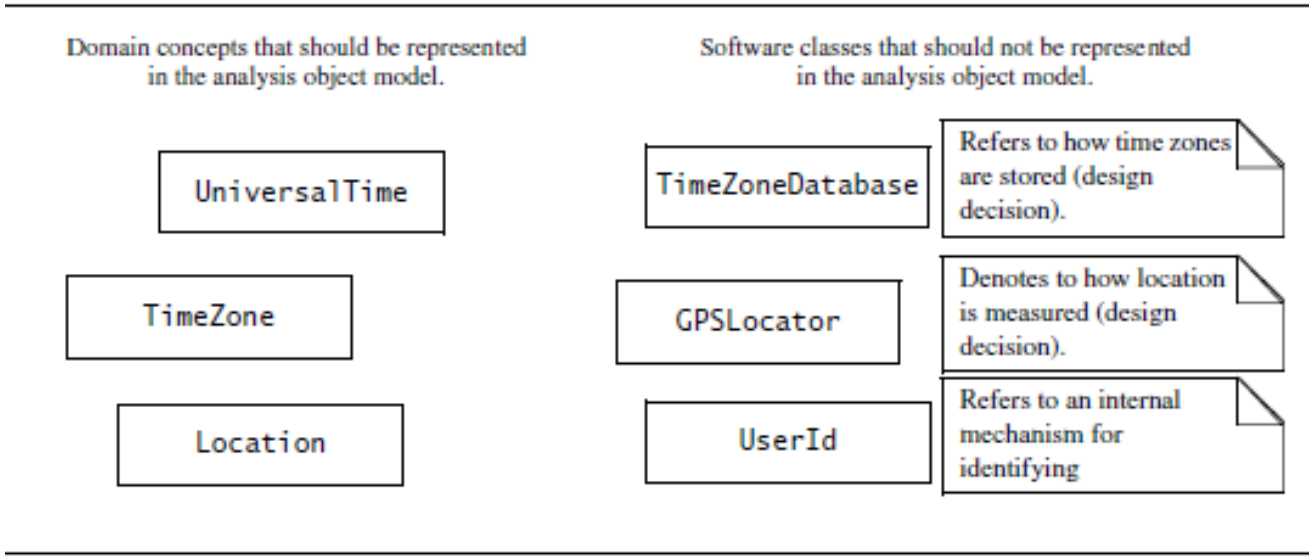
Figure 5-1 Ambiguity: what do you see?

An overview of OOSE development activities and their products



Analysis

When working with either the analysis object model or the dynamic model, it is essential to remember that these models represent user-level concepts, not actual software classes or components. For example, classes such as Database, Subsystem, SessionManager, Network, should not appear in the analysis model as the user is completely shielded from those concepts. Consequently, analysis classes should be viewed as
high-level abstractions that will be realized in much more detail later.



Activities during Object Modeling

Main goal: Find important abstractions

- Steps during object modeling
  1. Class identification
  2. Find attributes
  3. Find methods
  4. Find associations between classes
- Order of steps
  - Goal: get desired abstractions
  - Order of steps secondary, only a heuristic
- What happens if we find wrong abstractions?
  - We iterate and revise the model

Class Identification

Class identification is crucial to OO modeling

- Helps to identify important entities of a system

- Basic assumptions:

  1. We can find classes for a new software system (Forward Engineering)
  2. We can identify classes in an existing system (Reverse Engineering)

Class Identification is a Hard Problem

- One problem: Definition of system boundary:
  - Which abstractions are outside, which abstractions are inside the system boundary?
    - Actors are outside the system
    - Classes/Objects are inside the system.
- Another problem: Classes/Objects are not just found by taking a picture of a scene or domain
  - Application domain has to be analyzed
  - Depending on purpose of system, different objects might be found
    - How can we identify purpose of a system?
    - Scenarios and use cases => Functional model

Generalization vs Specialization

Generalization is the modeling activity that identifies abstract concepts from lower-level ones. For example, assume we are reverse-engineering an emergency management system and discover screens for managing traffic accidents and fires. Noticing common features among these three concepts, we create an abstract concept called Emergency to describe the common (and general) features of traffic accidents and fires.

Specialization is the activity that identifies more specific concepts from a high-level one. For example, assume that we are building an emergency management system from scratch a and that we are discussing its functionality with the client. The client first introduces us with theconcept of an incident, then describes three types of Incidents: Disasters, which require the collaboration of several agencies, Emergencies, which require immediate handling but can be handled by a single agency, and LowPriorityIncidents, that do not need to be handled if resources are required for other, higher-priority Incidents

Class Hierarchy

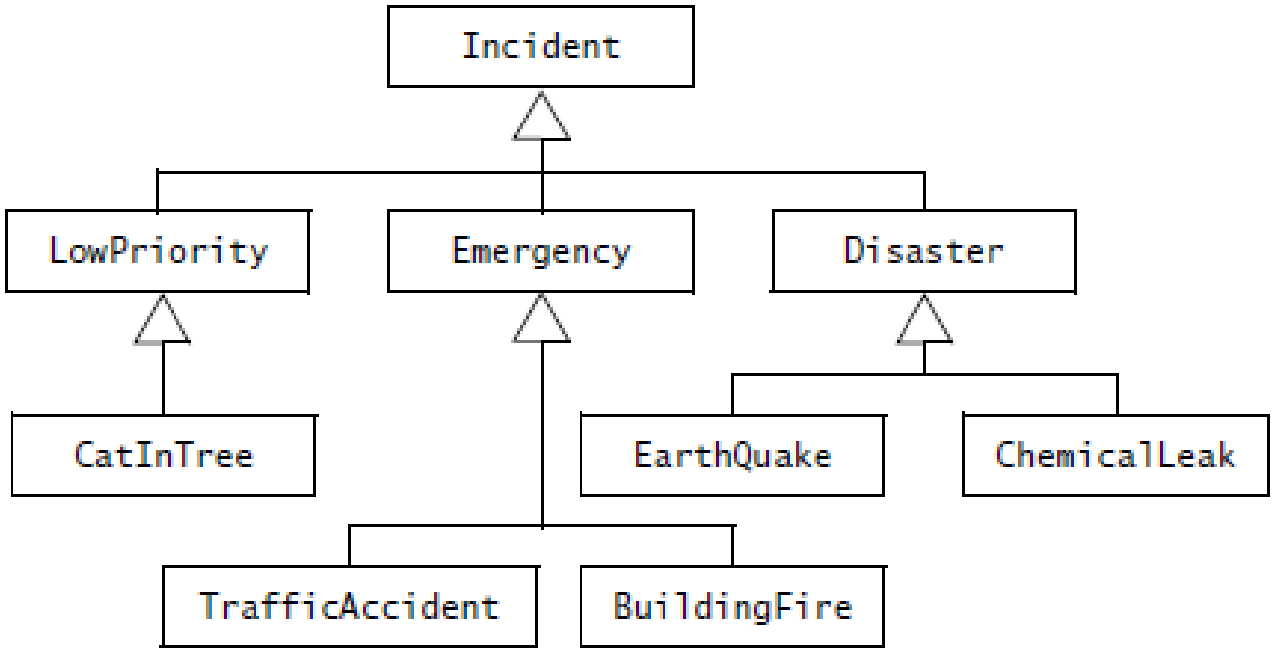


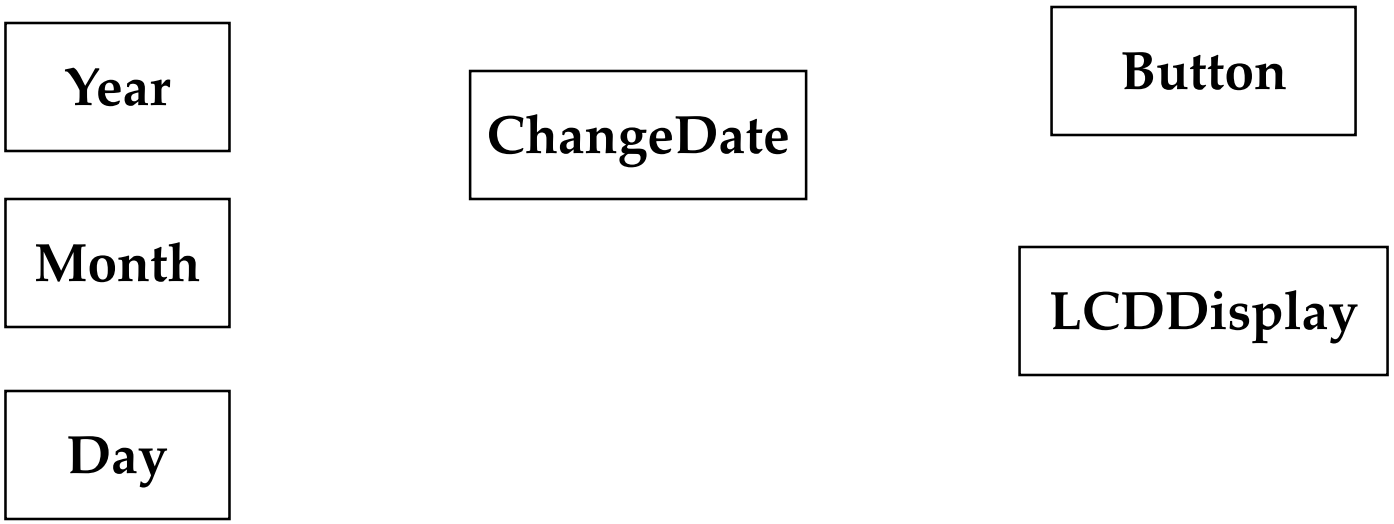
Figure 5-6 An example of a generalization hierarchy (UML class diagram). The top of the hierarchy represents the most general concept, whereas the bottom nodes represent the most specialized concepts.

There are different types of Objects

- Entity Objects
  - Represent persistent information tracked by the system (Application domain objects, also called "Business objects")
  - Objects representing system data, often from the domain model.
- Boundary Objects
  - Represent interaction between user and the system
  - Objects that interface with system actors (e.g. a **user** or **external service**). Windows, screens and menus are examples of boundaries that interface with users.
- Control Objects
  - Represent the control tasks performed by the system
  - Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.

Example: 2BWatch Modeling

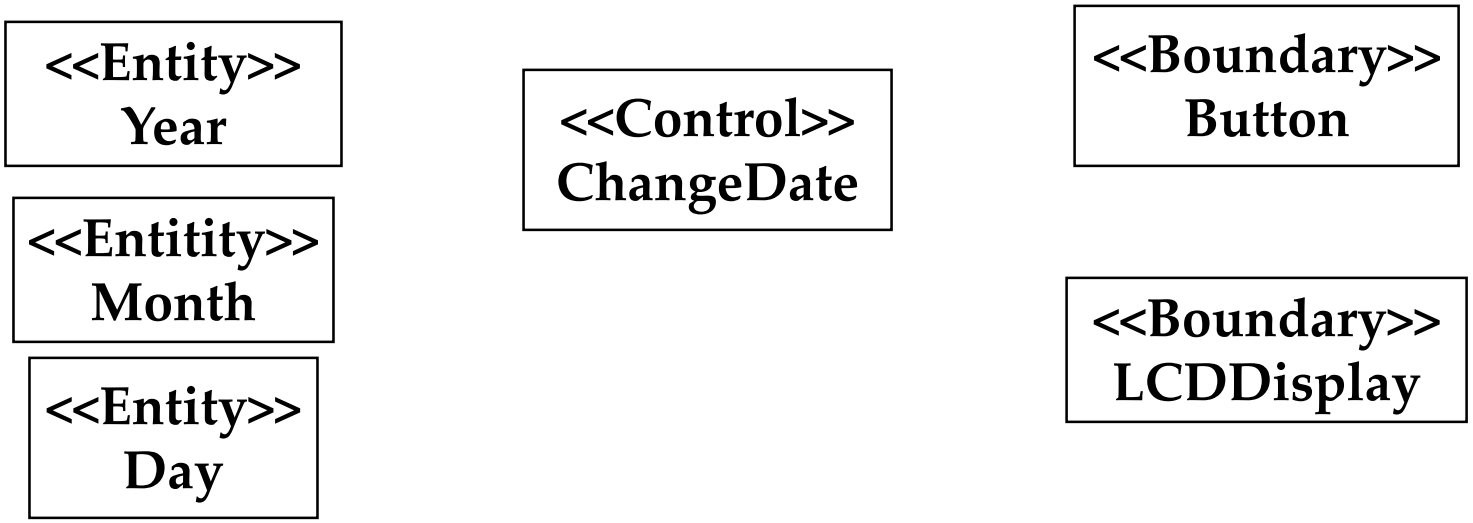
To distinguish different object types in a model we can use the UML Stereotype mechanism



Entity Objects Control Object Boundary Objects

Naming Object Types in UML

- UML provides stereotype mechanism to introduce new types of modeling elements
  - Drawn as a name enclosed by angled double-quotes (<<, >>) and placed before name of a UML element (class, method, attribute, …)
  - Notation: <<String>>Name

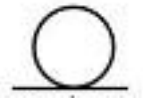


UML is an Extensible Language

- Stereotypes allow you to extend vocabulary of UML so that you can create new model elements, derived from existing ones.
- Example:
  - Stereotypes can also be used to classify method behavior such as <<constructor>>, <<getter>> or <<setter>>
- Stereotypes can be represented with icons and graphics:
  - This can increase readability of UML diagrams.

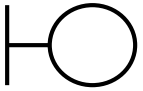
Icons for Stereotypes

- One can use icons to identify a stereotype
  - When the stereotype is applied to a UML model element, the icon is displayed beside or above the name

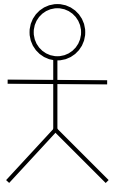


Year

ChangeDate



Button



WatchUser

Entity Object Control Object Boundary Object Actor

Pros and Cons of Stereotype Graphics

- Advantages:
  - UML diagrams may be easier to understand if they contain graphics and icons for stereotypes
    - This can increase the readability of the diagram, especially if the client is not trained in UML
    - And they are still UML diagrams!
- Disadvantages:
  - If developers are unfamiliar with the symbols being used, it can become much harder to understand what is going on
  - Additional symbols add to the burden of learning to read the diagrams.

Communication among objects

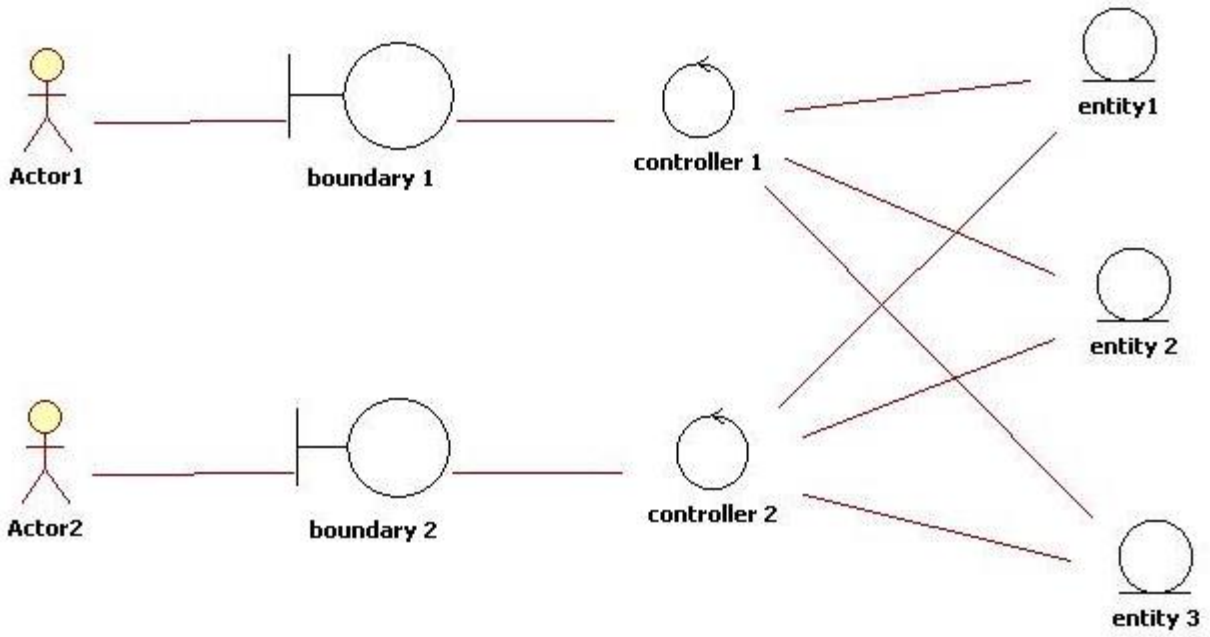
Four rules apply to their communication:

- Actors can only talk to boundary objects.
- Boundary objects can only talk to controllers and actors.
- Entity objects can only talk to controllers.
- Controllers can talk to boundary objects and entity objects, and to other controllers, but not to actor

Communication allowed:

	Entity	Boundary	Control
Entity	X		X
Boundary			X
Control	X	X	X

Communication among Objects



What is the benefit of Object types?

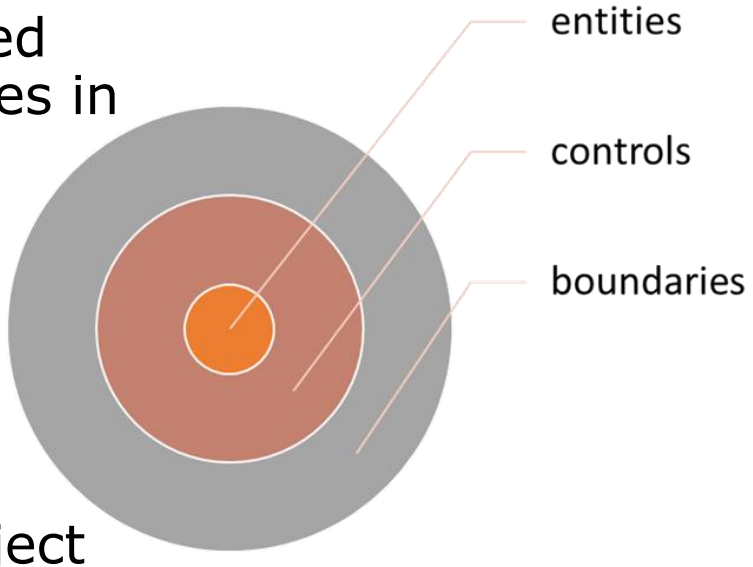
Object Types allow us to deal with Change

- Having three types of objects leads to models that are more resilient to change
  - Interface of a system changes more likely than control
  - The way the system is controlled changes more likely than entities in application domain
- Object types originated in Smalltalk:
  - Model, View, Controller (MVC)

Model <-> Entity Object

View <-> Boundary Object

Controller <-> Control Object



Finding Participating Objects in Use Cases

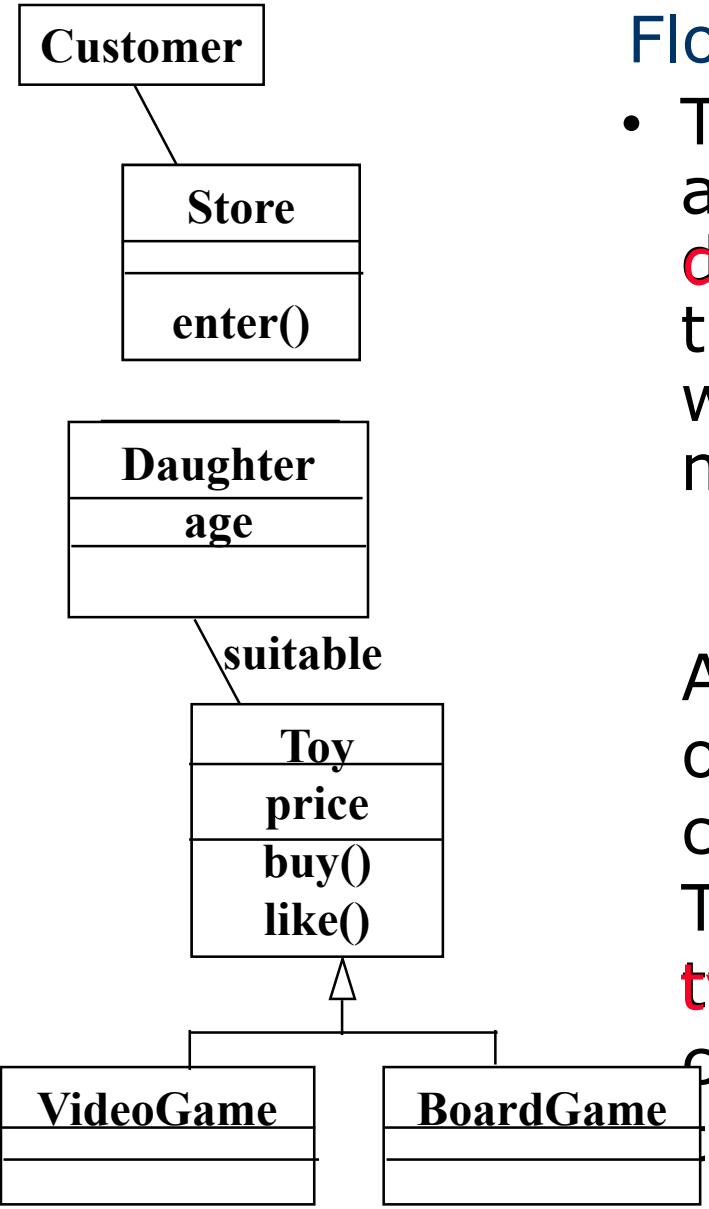
- Pick a use case and look at flow of events
- Do a textual analysis (noun-verb analysis)
  - Nouns are candidates for objects/classes
  - Verbs are candidates for operations
  - This is also called Abbott's Technique

- After objects/classes are found, identify their types
  - Identify real world entities that the system needs to keep track of (FieldOfficer Entity Object)
  - Identify real world procedures that the system needs to keep track of (EmergencyPlan Control Object)
  - Identify interface artifacts (PoliceStation Boundary Object).

Mapping parts of speech to model components (Abbot's Technique)

Example	*Part of speech*	*UML model component*
"Monopoly"	Proper noun	object
Toy	Improper noun	class
Buy, recommend	Doing verb	operation
is-a	being verb	inheritance
has an	having verb	aggregation
must be	modal verb	constraint
dangerous	adjective	attribute
enter	transitive verb	operation
depends on	intransitive verb	constraint, class, association

Generating a Class Diagram from Flow of Events



Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Ways to find Objects

- Syntactical investigation with Abbot's technique:
  - Flow of events in use cases
  - Problem statement

- Use other knowledge sources:
  - Application knowledge: End users and experts know the abstractions of the application domain
  - Solution knowledge: Abstractions in the solution domain
  - General world knowledge: Your generic knowledge and intution

Order of Activities for Object Identification

1. Formulate a few scenarios with help from an end user or application domain expert
2. Extract use cases from scenarios, with help of an application domain expert
3. Then proceed in parallel with following:
   - Analyze flow of events in each use case using Abbot's textual analysis technique
   - Generate UML class diagram

Steps in Generating Class Diagrams

1. Class identification (textual analysis, domain expert)
2. Identification of attributes and operations (sometimes before classes are found!)
3. Identification of relations:
   - Associations between classes
   - Multiplicities
   - Inheritance

Who uses Class Diagrams?

- Purpose of class diagrams
  - Description of static properties of a system
- Main users of class diagrams:
  - Application domain expert
    - uses class diagrams to model application domain (including taxonomies)
      - during requirements elicitation and analysis
  - Developer
    - uses class diagrams during development of a system
      - during analysis, system design, object design and implementation

Who does not use Class Diagrams?

- The client and the end user are usually not interested in class diagrams
  - Clients focus more on project management issues
  - End users are more interested in functionality of the system.

Developers have different Views on Class Diagrams

- According to the development activity, a developer plays different roles:
  - Analyst
  - System Designer
  - Object Designer
  - Implementor /Development Engineer

- Each of these roles has a different view about class diagrams (object model).

The View of the Analyst

- The analyst is interested
  - in application classes: associations between classes are relationships between abstractions in application domain
  - operations and attributes of application classes
- The analyst uses inheritance in the model to reflect taxonomies in application domain
    - Taxonomy: An is-a-hierarchy of abstractions in an application domain
- The analyst is not interested
  - In exact signature of operations
  - in solution domain classes

The View of the Designer

- The designer focuses on solution of problem (i.e., solution domain)
- Associations between classes are now references (pointers) between classes in solution domain
- The designer describes interface of subsystems (system design) and classes (object design)

Goals of the Designer

- Design usability: interfaces are usable from as many classes as possible within the system
- Design reusability: interfaces are designed in a way that they can also be reused by other (future) software systems
  - => Class libraries
  - => Frameworks
  - => Design patterns

The View of the Implementor /Dev. Engineer

- Class implementor
  - Must realize interface of a class in a programming language
  - Interested in appropriate data structures (for attributes) and algorithms (for operations)
- Class extender
  - Interested in how to extend a class to solve a new problem or to adapt to a change in application domain
- Class user
  - Interested in signatures of class operations and conditions, under which they can be invoked
  - NOT interested in implementation of a class

Why distinguish Users of Class Diagrams?

- Models often don't distinguish between application classes and solution classes
  - Reason: Modeling languages like UML allow use of both types of classes in same model
    - "address book", "array"
  - Preferred: No solution classes in analysis model
- Many systems don't distinguish between specification and implementation of a class
  - Reason: Object-oriented programming languages allow simultaneous use of specification and implementation of a class
  - Preferred: We distinguish between analysis model and object design model. Analysis model does not contain any implementation specification.

Analysis Model vs Object Design model

- Analysis model constructed during analysis phase
  - Main stake holders: End user, customer, analyst
  - Class diagrams contain only application domain classes

- Object design model (sometimes also called specification model) created during object design phase
  - Main stake holders: class specifiers, class implementors, class users and class extenders
  - Class diagrams contain application domain as well as solution domain classes

Summary

- System modeling
  - Functional + object + dynamic modeling
- Functional modeling
  - From scenarios to use cases to objects
- Object modeling is the central activity
  - Class identification is a major activity of object modeling
  - Easy syntactic rules to find classes and objects (Abbot's Technique)

Summary

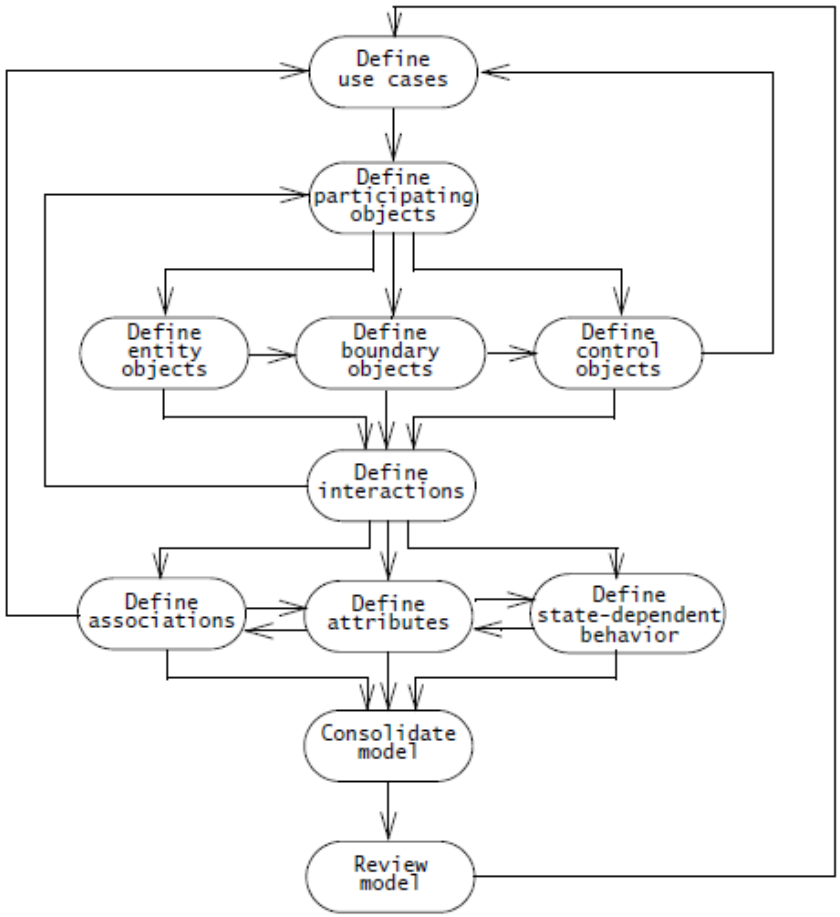


Figure 5-19 Analysis activities (UML activities diagram).