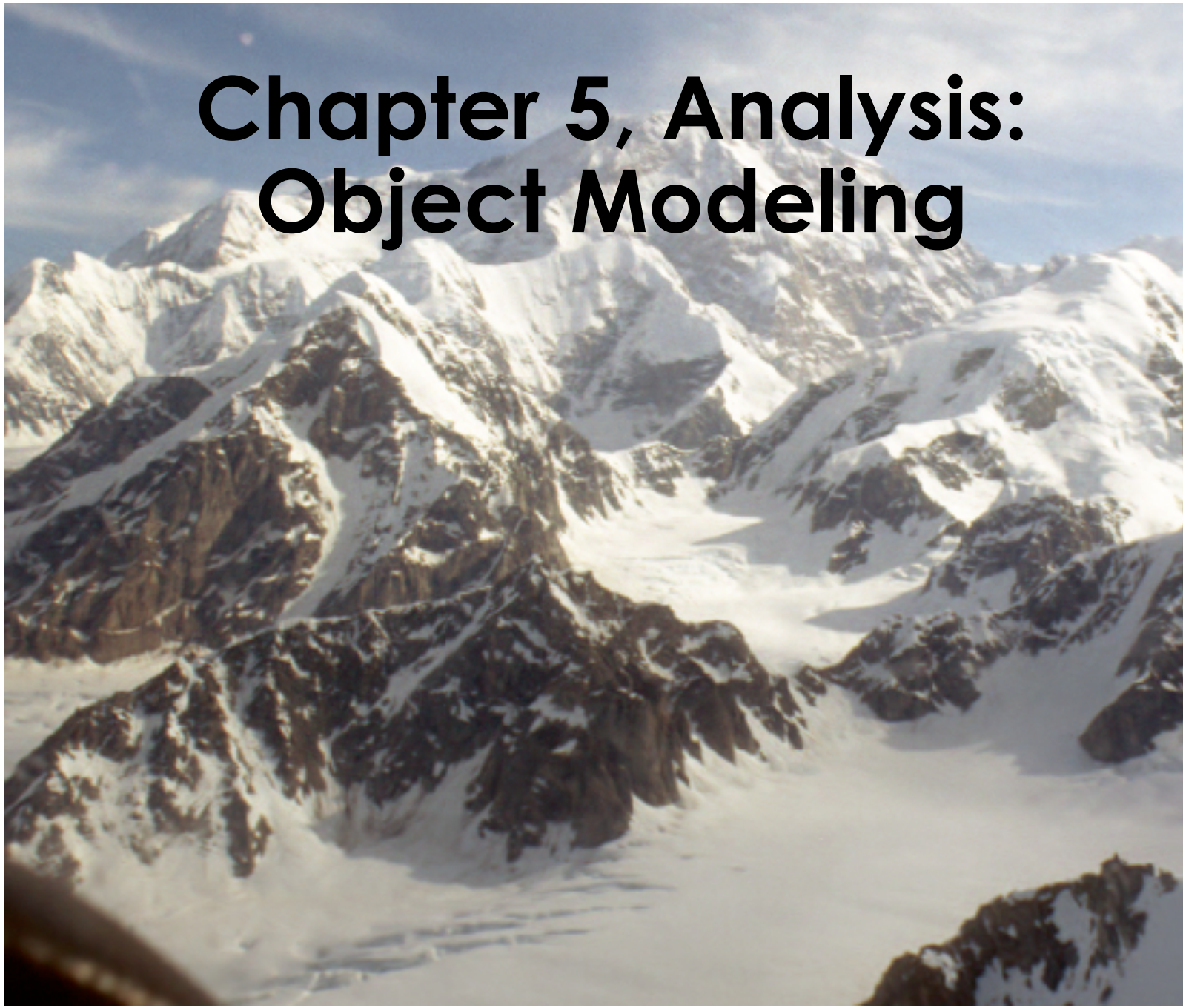**Object-Oriented Software Engineering**
Using UML, Patterns, and Java

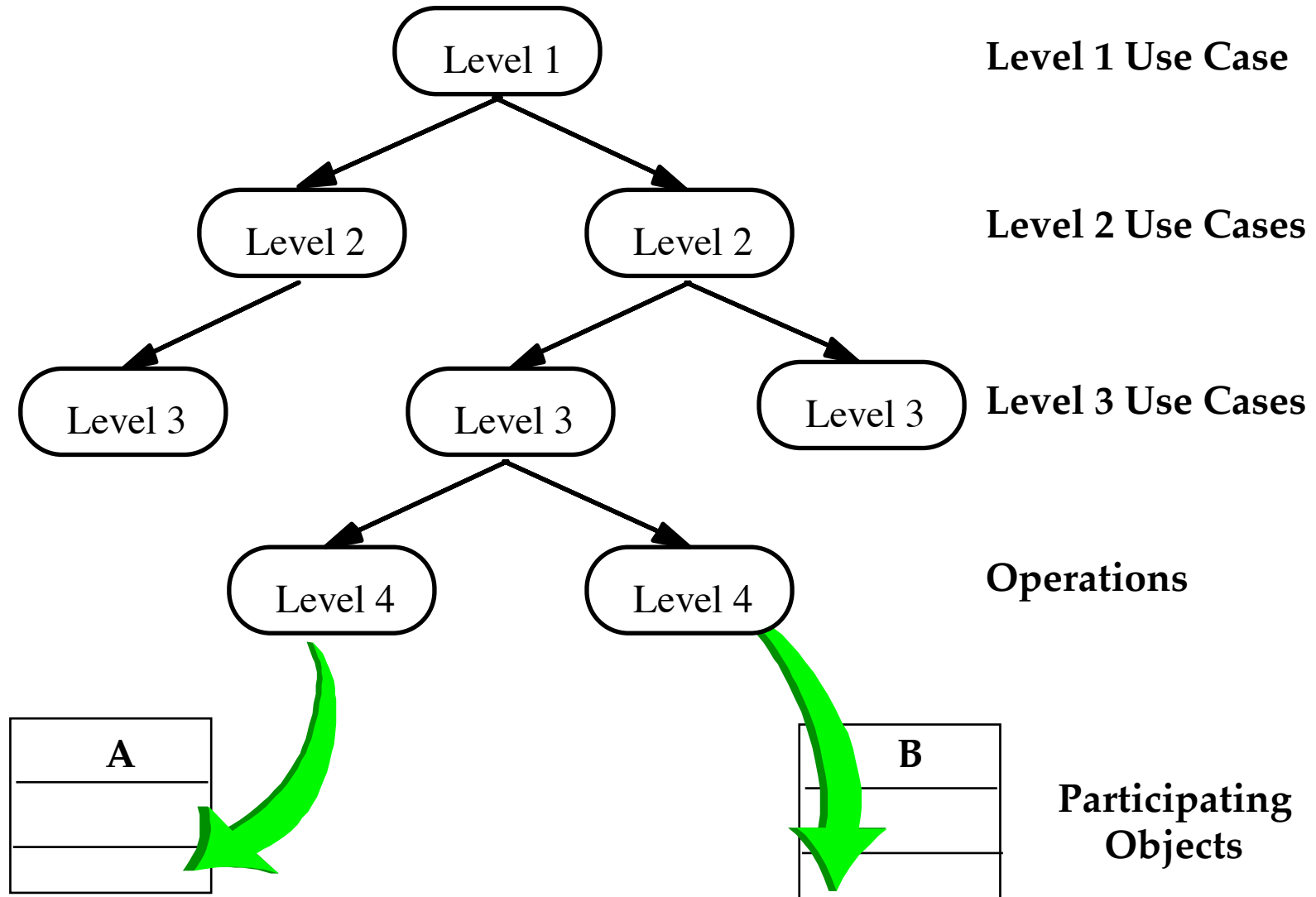# Chapter 5, Analysis: Object Modeling

Outline

Recall: System modeling = Functional modeling + Object modeling + Dynamic modeling
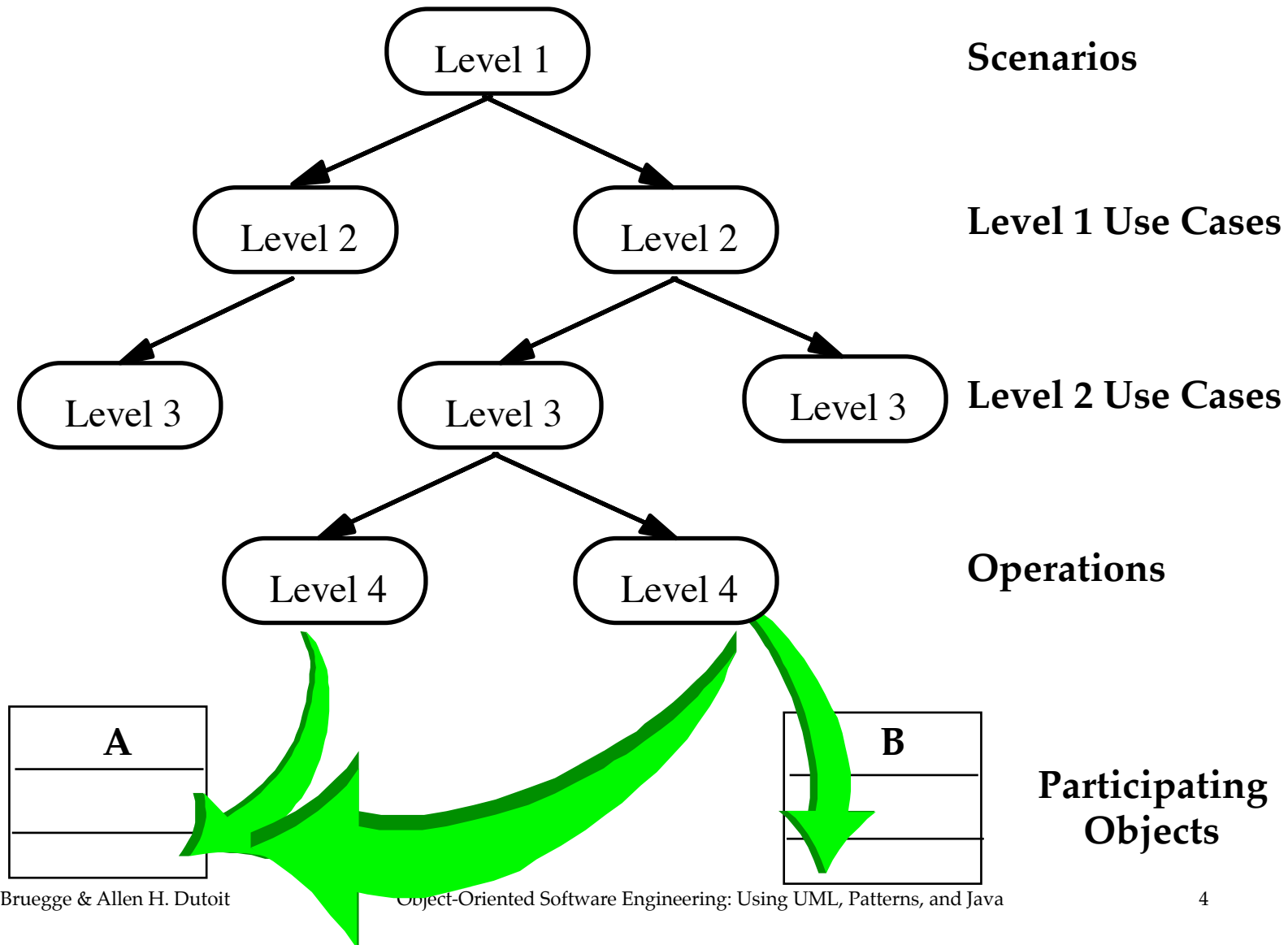
✓ Last lecture: Functional modeling

- Now: Object modeling
  - Activities during object modeling
  - Object identification
  - Object types
    - Entity, boundary and control objects
  - Abott's technique
    - Helps in object identification.

From Use Cases to Objects

Level 1 — **Level 1 Use Case**

Level 2   Level 2 — **Level 2 Use Cases**

Level 3   Level 3   Level 3 — **Level 3 Use Cases**

Level 4   Level 4 — **Operations**

A   B — **Participating Objects**

From Use Cases to Objects: Why Functional Decomposition is not Enough

Level 1   **Scenarios**

Level 2   Level 2   **Level 1 Use Cases**

Level 3   Level 3   Level 3   **Level 2 Use Cases**

Level 4   Level 4   **Operations**

A   B   **Participating Objects**

Activities during Object Modeling

Main goal: Find the important abstractions

- Steps during object modeling
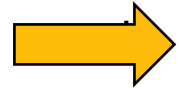
  ➡ Class identification

  - Based on the fundamental assumption that we can find abstractions

  2. Find the attributes

  3. Find the operations

  4. Find the associations between classes

- Order of steps

  - Goal: get the desired abstractions

  - Order of steps is secondary

- What happens if we find the wrong abstractions?

  - We iterate and revise the model.

Class Identification

Class identification is crucial to object-oriented modeling

- Helps to identify the important entities of a system

- Basic assumptions:

  1. We can find the classes for a new software system (Forward Engineering)

  2. We can identify the classes in an existing system (Reverse Engineering)

- Why can we do this?

  - Philosophy, science, experimental evidence.

Class Identification

- Approaches
  - Application domain approach
    - Ask application domain experts to identify relevant abstractions
  - Syntactic approach
    - Start with use cases
    - Analyze the text to identify the objects
    - Extract participating objects from flow of events
  - Design patterns approach
    - Identify relevant abstractions that can be reused (apply design knowledge)
  - Component-based approach
    - Identify existing solution classes.

Class identification is a Hard Problem

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

There are different types of Objects

- ## Entity Objects
  - Represent the persistent information tracked by the system (Application domain objects, also called "Business objects")

- ## Boundary Objects
  - Represent the interaction between the user and the system

- ## Control Objects
  - Represent the control tasks performed by the system.

From Use Cases to Objects

Starting from use cases and scenarios, analysis activities performed to obtain the analysis model are:

- Identifying entity objects
- Identifying boundary objects
- Identifying control objects
- Mapping use cases to objects
- Identifying associations among objects
- Identifying object attributes
- Modeling behavior with statecharts
- Modeling generalization relationships
- Reviewing the analysis model

Example: 2BWatch Modeling

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

Year

ChangeDate

Button

Month

LCDDisplay

Day

Entity Objects   Control Object   Boundary Objects

Naming Object Types in UML

| <<Entity>> Year |
| <<Entity>> Month |
| <<Entity>> Day |

| <<Control>> ChangeDate |

| <<Boundary>> Button |

| <<Boundary>> LCDDisplay |

Entity Object   Control Object   Boundary Object

Icons for Object Types: Entity, Control and Boundary Object

- We can also 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

System Boundary

| Year	ChangeDate	Button	WatchUser
Entity Object	Control Object	Boundary Object	Actor
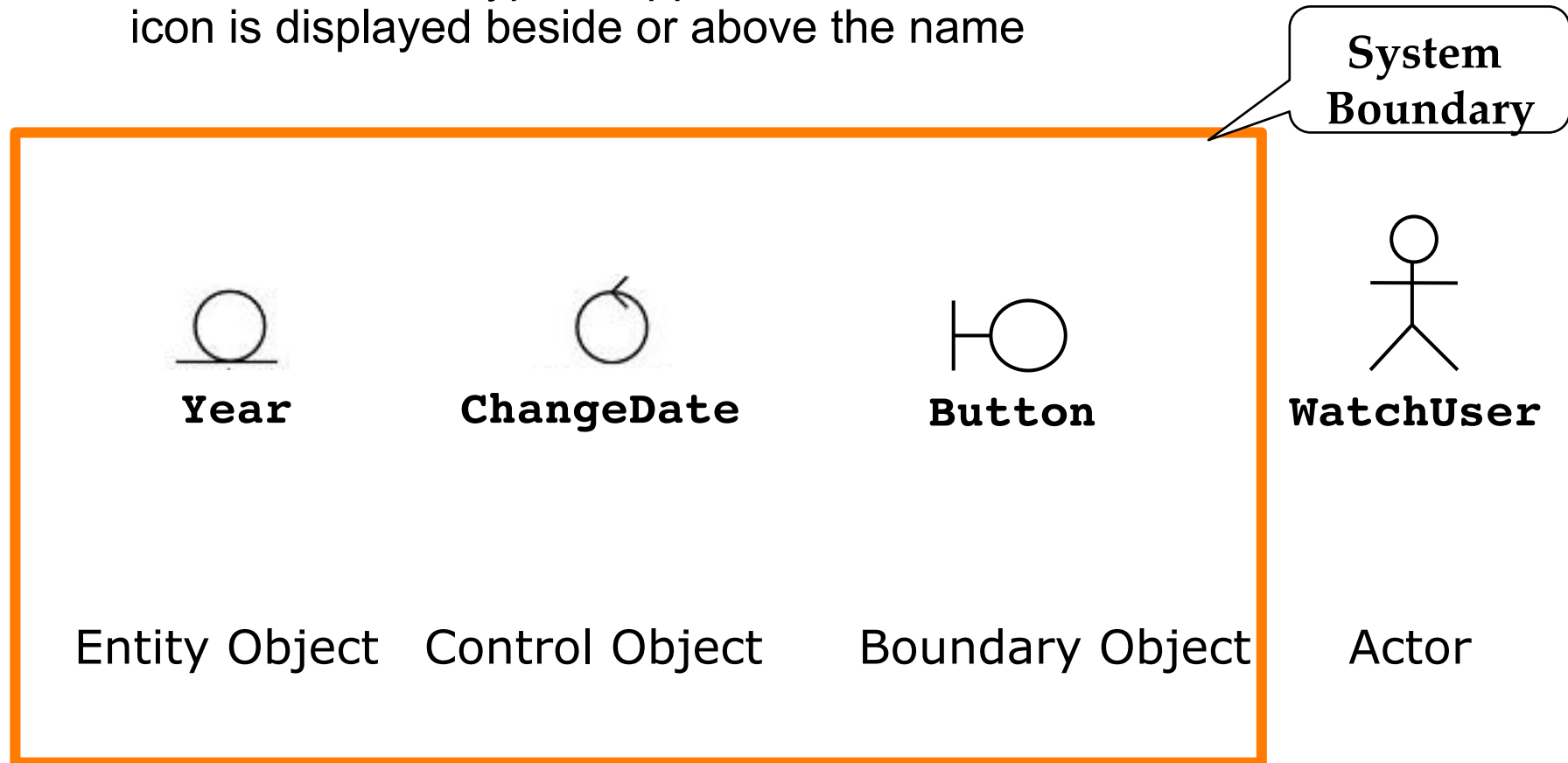
UML is an Extensible Language

- Stereotypes allow you to extend the vocabulary of the UML so that you can create new model elements, derived from existing ones

- Other Examples:
  - Stereotypes can also be used to classify method behavior such as <<constructor>>, <<getter>> or <<setter>>
  - To indicate the interface of a subsystem or system, one can use the stereotype <<interface>> (Lecture System Design)

- Stereotypes can be represented with icons as well as graphics:
  - This can increase the readability of UML diagrams.

Graphics for Stereotypes

- One can also use graphical symbols to identify a stereotype
  - When the stereotype is applied to a UML model element, the graphic replaces the default graphic for the diagram element.

- Example: When modeling a network, we can define graphical symbols to represent classes of type Switch, Server, Router and Printer.

Graphics for Class of type Router

Graphics for Class of type Switch

Graphics for Server Class

Bernd Bruegg
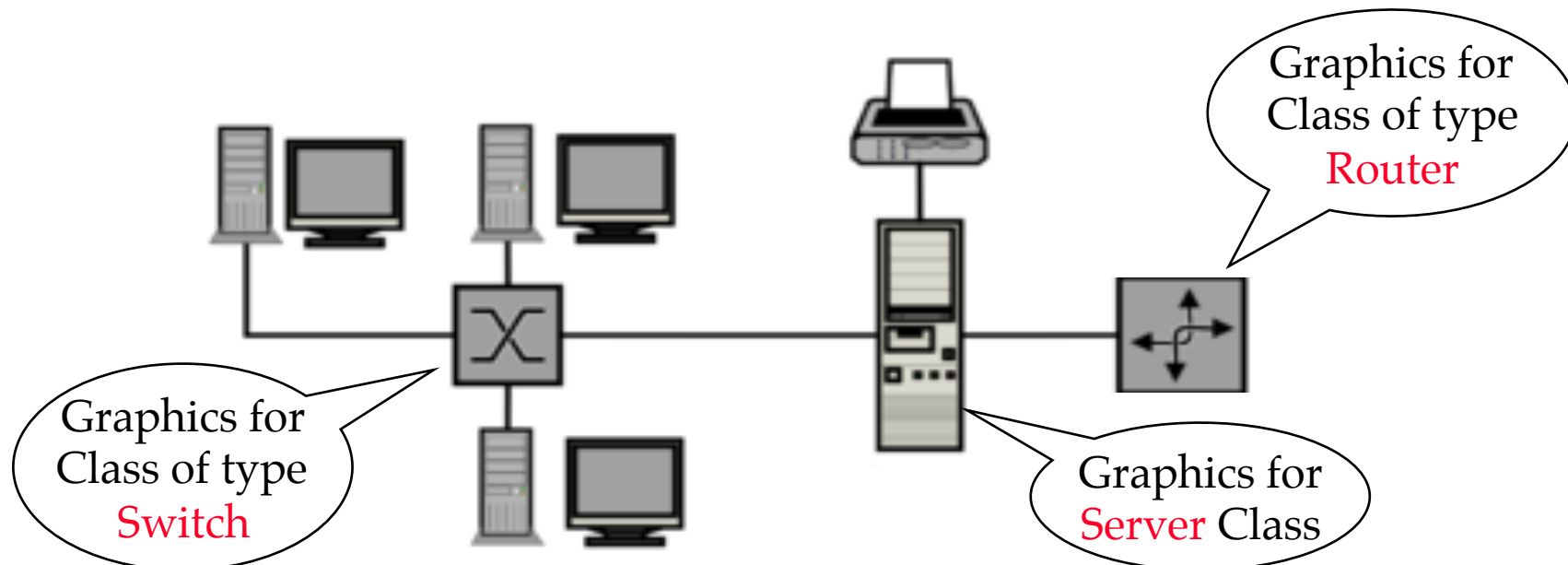
Pros and Cons of Stereotype Graphics

- 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.

Object Types allow us to deal with Change

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

          Model   <-> Entity Object
           View <-> Boundary Object
        Controller <-> Control Object

- Next topic: Finding objects.

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 object model components [Abbot 1983]

Part of speech	Model component
Proper noun	object
Improper noun	class
Doing verb	method
being verb	inheritance
having verb	aggregation
modal verb	constraint
adjective	attribute
transitive verb	method
intransitive verb	method (event)

Example for using the Technique

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

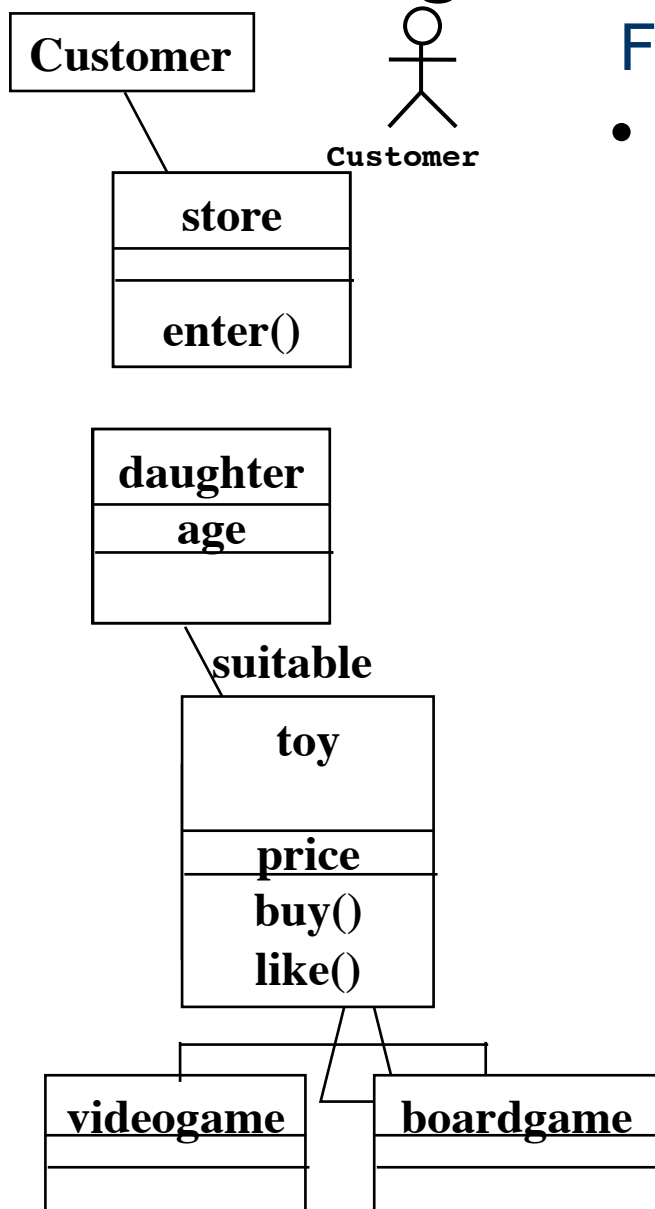The suitability of the game depends on the age of the child

His daughter is 3 years old

The assistant recommends another type of toy, the boardgame "Monopoly".

Mapping grammatical constructs to model components (Abbot's Technique)

Example	Grammatical construct	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

```
┌─────────────┐
│  Customer   │
└─────────────┘
```

Customer

```
┌─────────────┐
│    store    │
├─────────────┤
│             │
├─────────────┤
│   enter()   │
└─────────────┘
```

```
┌─────────────┐
│  daughter   │
├─────────────┤
│    age      │
├─────────────┤
│             │
└─────────────┘
```

suitable

```
┌─────────────┐
│    toy      │
├─────────────┤
│   price     │
├─────────────┤
│   buy()     │
│   like()    │
└─────────────┘
```

```
┌──────────────┐   ┌──────────────┐
│  videogame   │   │  boardgame   │
├──────────────┤   ├──────────────┤
│              │   │              │
├──────────────┤   ├──────────────┤
│              │   │              │
└──────────────┘   └──────────────┘
```

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 the help from an end user or application domain expert

2. Extract the use cases from the scenarios, with the help of an application domain expert

3. Then proceed in parallel with the following:

   - Analyse the flow of events in each use case using Abbot's textual analysis technique

   - Generate the UML class diagram.

Steps in Generating Class Diagrams

1. Class identification (textual analysis, domain expert)
2. Identification of attributes and operations (sometimes before the classes are found!)
3. Identification of associations between classes
4. Identification of multiplicities
5. Identification of roles
6. Identification of inheritance

How to identify entity objects

- terms that developers or users need to clarify in order to understand the use case

- recurring nouns in the use cases (e.g., Incident)

- real-world entities that the system needs to keep track of (e.g., FieldOfficer, Dispatcher, Resource)

- real-world activities that the system needs to keep track of (e.g., Emergencyoperationsplan)

- use cases (e.g., ReportEmergency)

- data sources or sinks (e.g., Printer)

- always use the user's terms

How to identify boundary objects

- Identify forms and windows the users needs to enter data into the system (e.g., EmergencyReportForm, ReportEmergencyButton).

- Identify notices and messages the system uses to respond to the user (e.g., AcknowledgmentNotice).

- Do not model the visual aspects of the interface with boundary objects (user mock-ups are better suited for that).

- Always use the user's terms for describing interfaces as opposed to terms from the implementation technology.

How to identify control objects

- Identify one control object per use case or more if the use case is complex and if it can be divided into shorter flows of events.

- Identify one control object per actor in the use case.

- The life span of a control object should be extent of the use case or the extent of a user session. If it is difficult to identify the beginning and the end of a control object activation, the corresponding use case may not have a well-defined entry and exit condition.

Attributes

- Detection of attributes is application specific
- Attributes in one system can be classes in another system
- Turning attributes to classes and vice versa

Associations

- Types of Associations
  - Canonical associations
    - Part-of Hierarchy (Aggregation)
    - Kind-of Hierarchy (Inheritance)
  - Generic associations

Operations

- Source of operations
    - Use cases in the functional model
    - General world knowledge
    - Generic operations: Get/Set
    - Design Patterns
    - Application domain specific operations
    - Actions and activities in the dynamic model

An example: the ReportEmergency use case

Use case name: ReportEmergency

Entry condition: The FieldOfficer activates the "Report Emergency" function of her terminal.

Flow of events:

1. FRIEND responds by presenting a form to the officer. The form includes an emergency type menu (Genera1 emergency, fire, transportation), a location, incident description, resource request, and hazardous material fields.

2. The FieldOfficer fills the form, by specifying minimally the emergency type and description fields. The FieldOfficer may also describes possible responses to the emergency situation and request specific resources. Once the form is completed, the FieldOfficer submits the form by pressing the "Send Report" button, at which point, the Dispatcher is notified.

3. The Dispatcher reviews the information submitted by the FieldOf f icer and creates an incident in the database by invoking the OpenIncident use case. Al1 the information contained in the FieldOfficer's form is automatically included in the incident. The Dispatcher selects a response by allocating resources to the incident (with the AllocateResources use case) and acknowledges the emergency report by sending a FRIENDgram to the FieldOfficer.

Exit condition: The FieldOfficer receives the acknowledgment and the selected response.

Entity objects in the example

Dispatcher

Police officer who manages Incidents. A Dispatcher opens, documents, and closes Incidents in response to Emergency Reports and other communication with FieldOfficers. Dispatchers are identified by badge numbers.

EmergencyReport

Initial report about an Incident from a FieldOfficer to a Dispatcher. An EmergencyReport usually triggers the creation of an Incident by the Dispatcher. An EmergencyReport is composed of a emergency level, a type (fire, road accident, or other), a location, and a description.

Fieldofficer

Police or fire officer on duty. A FieldOfficer can be allocated to, at most, one Incident at a time. FieldOfficers are identified by badge numbers.

NOT TRUE in this UC. Here FieldOfficers are actors, they can be entity objects in the Allocateresource UC

Incident

Situation requiring attention from a FieldOfficer. An Incident may be reported in the system by a FieldOfficer or anybody else external to the system. An Incident is composed of a description, a response, a status (open, closed, documented), a location, and a number of FieldOfficers.

Boundary Objects in the example

AcknowledgmentNotice

Notice used for displaying the Dispatcher's acknowledgment to the FieldOfficer.

DispatcherStation

Computer used by the Dispatcher.

ReportEmergencyButton

Button used by a FieldOfficer to initiate the ReportEmergency use case.

EmergencyReportForm

Form used for the input of the ReportEmergency. This form is presented to the FieldOfficer on the FieldOfficerstation when the "Report Emergency" function is selected. The EmergencyReportForm contains fields for specifying all attributes of an emergency report and a button (or other control) for submitting the form once it is completed.

FieldOfficerStation

Mobile computer used by the FieldOfficer.

Incident Form

Form used for the creation of Incidents. This form is presented to the Dispatcher on the DispatcherStation when the EmergencyReport is received. The Dispatcher also uses this form to allocate resources and to acknowledge the FieldOfficer's report.

Control objects in the example

ReportEmergencyControl

Manages the report emergency reporting function on the FieldOfficerstation. This object is created when the FieldOfficer selects the "Report Emergency" button. It then creates an EmergencyReportFom and presents it to the FieldOfficer. After submission of the form, this object then collects the information from the form, creates an EmergencyReport, and forwards it to the Dispatcher. The control object then waits for an acknowledgment to come back from the DispatcherStation. When the acknowledgment is received, the ReportEmergencyControl object creates an AcknowledgmentNotice and displays it to the Fie1dOfficer .

ManageEmergencyControl

Manages the report emergency reporting function on the DispatcherStation. This object is created when an EmergencyReport is received. It then creates an IncidentFom and displays it to the Dispatcher. Once the Dispatcher has created an Incident, allocated Resources, and submitted an acknowledgment, ManageEmergencyControl forwards the acknowledgment to the FieldOfficerstation.

Who uses Class Diagrams?

- Purpose of class diagrams
  - The description of the static properties of a system

- The main users of class diagrams:
  - The application domain expert
    - uses class diagrams to model the application domain (including taxonomies)
      - during requirements elicitation and analysis
  - The developer
    - uses class diagrams during the 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 the functionality of the system.

Summary

- System modeling
  - Functional modeling+object modeling+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
- Class diagrams are the "center of the universe" for the object-oriented developer
  - The end user focuses more on the functional model and and usability.