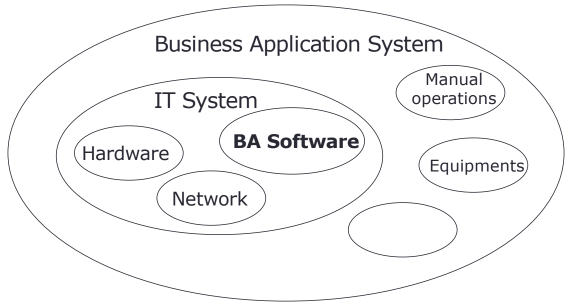
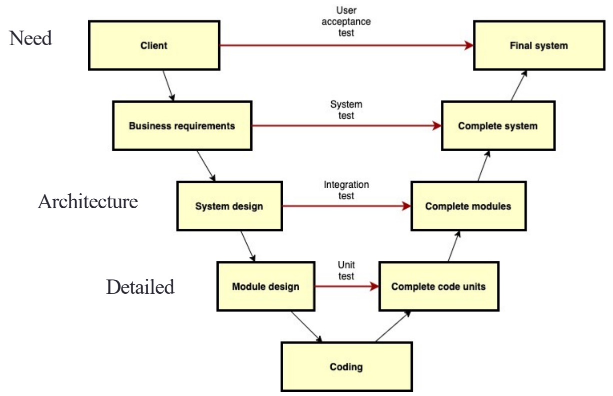
A big picture of Business Application System



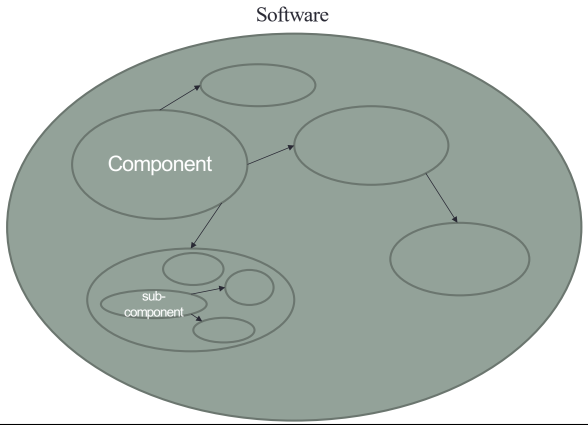
1

V-model

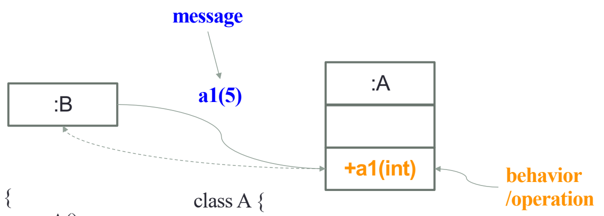


2

Procedural (e.g. C): Function, Procedure
Object-Oriented (e.g. Java): Class (, Subsystem)



3



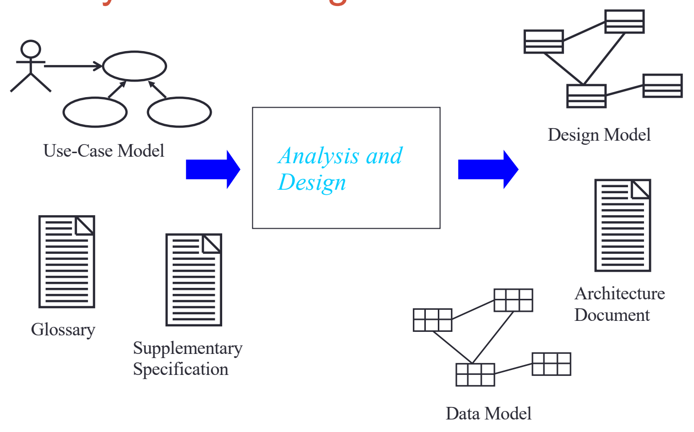
```
class B {
    A a = new A();

    +b(){
        a.a1(5);
    }
}
```

```
class A {
        - attr1;
        - attr2;
        - attr3;
        - attr4;

        +a1(int i){
                //tính tiền trong ví
                //tính tiền trong tài khoản…
                //trả về tổng số tiền
        }

}
```

method

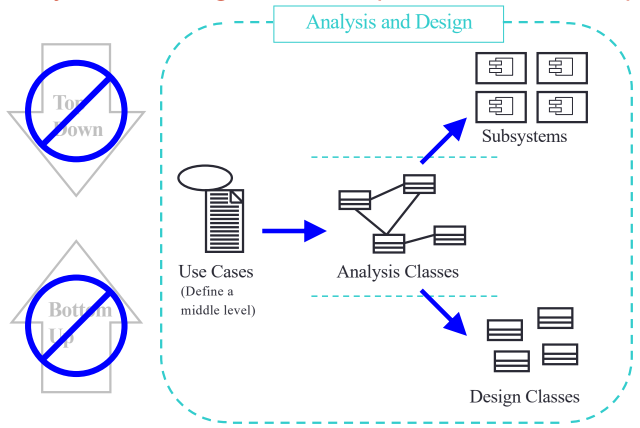
4

Analysis and Design Overview



5

Analysis and Design Are Not Top-Down or Bottom-Up



6

ITSS SOFTWARE DEVELOPMENT/SOFTWARE DESIGN AND CONSTRUCTION

3. USE CASE ANALYSIS

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn

Some slides extracted from IBM coursewares

7

Content

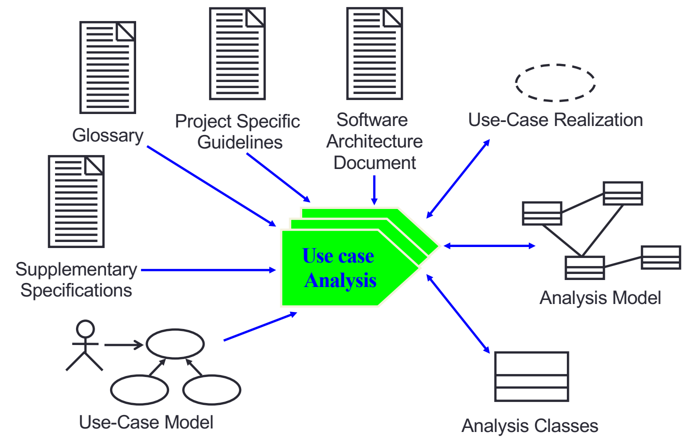
8

Review: Software Architectural Design process

- Purpose: "to provide a design for the software that implements and can be verified against the requirements"
- Software architecture is designed from the software requirements
- Main items
  - a top-level structure of the software and the software components which constructs the software
  - a top-level design for the interfaces external to the software and between the software components
  - a top-level design for the database

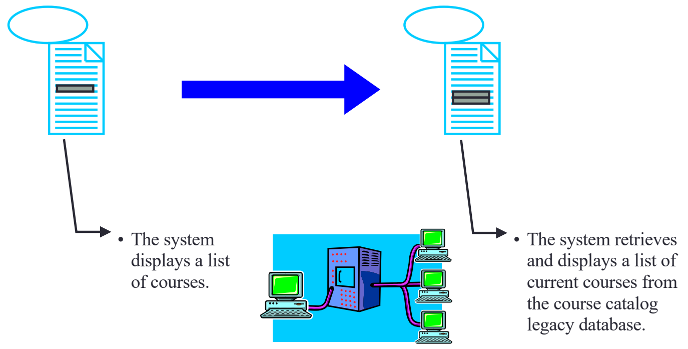
9

Use case analysis Overview



10

Supplement the Use-Case Specification



- The system displays a list of courses.

- The system retrieves and displays a list of current courses from the course catalog legacy database.

11

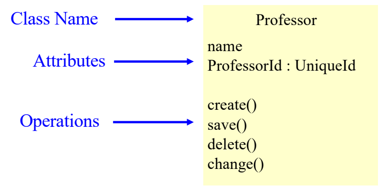
Content

1. Overview
2. Analysis classes
3. Distribute Use-Case Behavior to Classes
4. Analysis class diagram

12

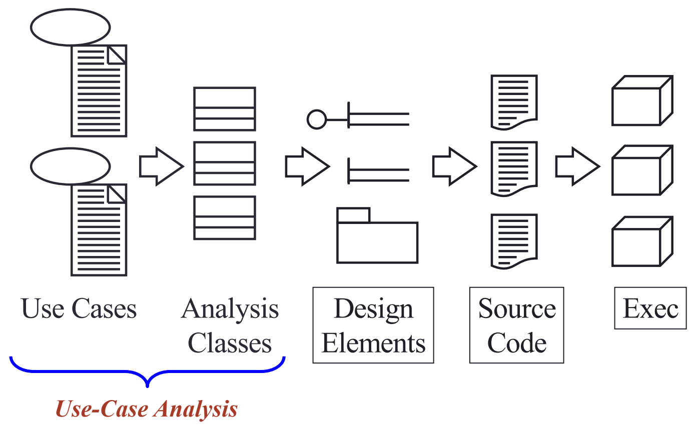
Review: Class

- An abstraction
- Describes a group of objects with common:
  - Properties (attributes)
  - Behavior (operations)
  - Relationships
  - Semantics



13

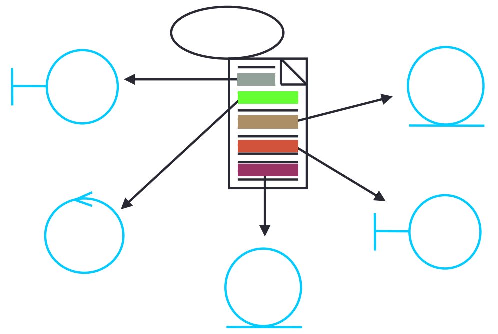
Analysis Classes: A First Step Toward Executables



14

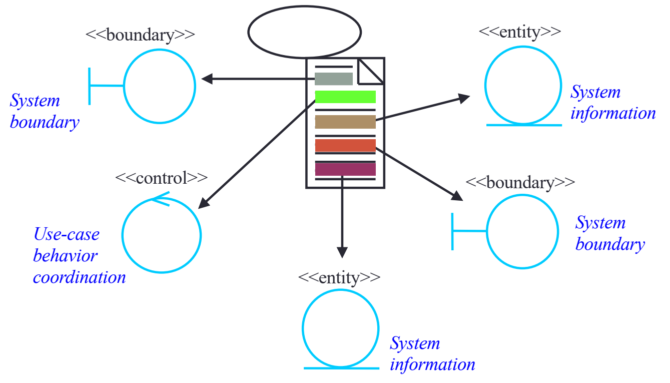
Find Classes from Use-Case Behavior

- The complete behavior of a use case has to be distributed to analysis classes



15

Types of Analysis Classes

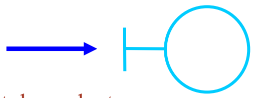


16

2.1. Boundary Classes

- Intermediate between the interface and something outside the system
- Several Types
  - User interface classes
  - System interface classes
  - Device interface classes
- One boundary class per actor/use-case pair (typical)

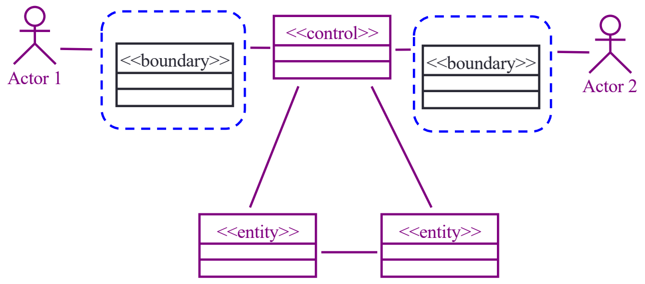
Analysis class stereotype



Environment dependent.

17

The Role of a Boundary Class

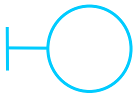


Model interaction between the system and its environment.

18

Example in AIMS: Finding Boundary Classes for UC "Place order" and "Pay order"

- Find boundary classes per actor/use case pair
  - Typical one



19

Guidelines: Boundary Classes

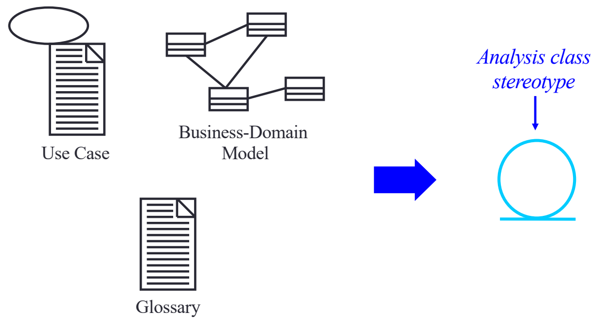
- User Interface Classes
  - Concentrate on what information is presented to the user
  - Do NOT concentrate on the UI details
- System and Device Interface Classes
  - Concentrate on what protocols must be defined
  - Do NOT concentrate on how the protocols will be implemented

Concentrate on the responsibilities, not the details!

20

2.2. Entity Classes

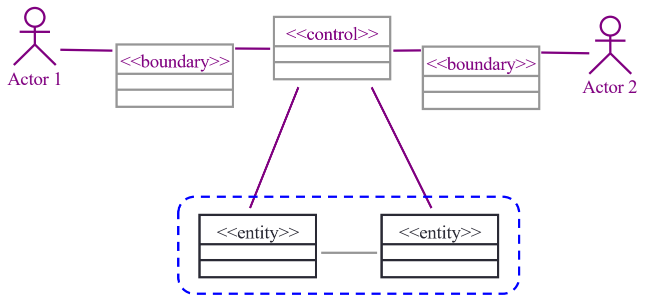
- Key abstractions of the system



Environment independent.

21

The Role of Entity Classes



Store and manage information in the system.

22

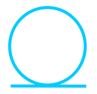
Guidelines: Entity Classes

- Use use-case flow of events as input
- Key abstractions of the use case
- Traditional, filtering nouns approach
  - Underline noun clauses in the use-case flow of events
  - Remove redundant candidates
  - Remove vague candidates
  - Remove actors (out of scope)
  - Remove implementation constructs
  - Remove attributes (save for later)
  - Remove operations

23

Example in AIMS: Finding Entity Classes for UC "Place order" and "Pay order"

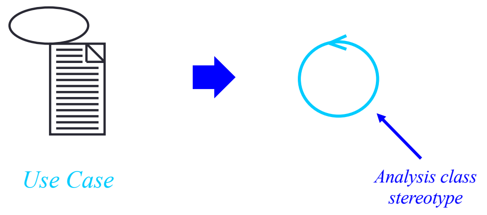
- Find candidate entity classes



24

3.3. Control Classes

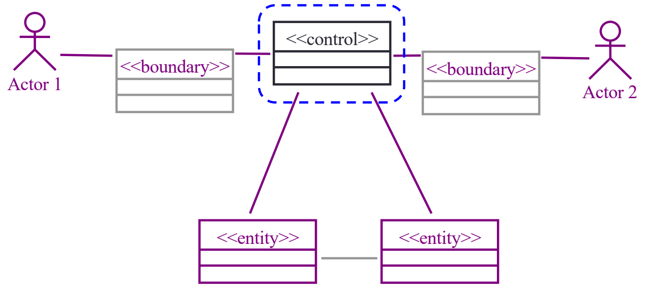
- ◆Provide coordinating behavior in the system
- ◆model control behavior specific to one or more use cases



Use-case dependent. Environment independent.

25

The Role of Control Classes



Coordinate the use-case behavior.

26

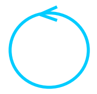
Guidelines: Control Classes

- ◆In general, identify one control class per use case.
- ◆The system can perform some use cases without control classes by using just entity and boundary classes.
  - This is particularly true for use cases that involve only the simple manipulation of stored information.
- ◆More complex use cases generally require one or more control classes to coordinate the behavior of other objects in the system.
  - Examples of control classes include transaction managers, resource coordinators, and error handlers.

27

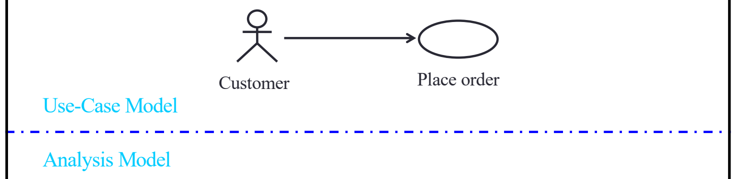
Example in AIMS: Finding Control Classes for UC "Place order" and "Pay order"

- One control class per use case (typical)



28

Course AIMS Summary: Analysis Classes



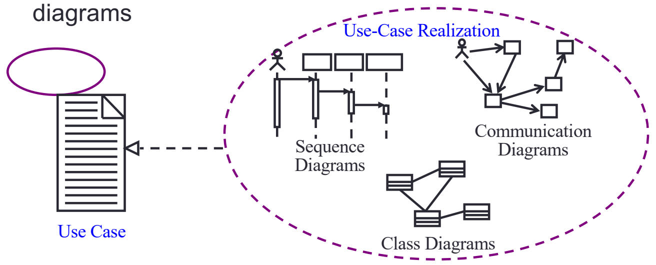
29

Content

30

3. Distribute Use-Case Behavior to Classes

- For each use-case flow of events:
  - Identify analysis classes
  - Allocate use-case responsibilities to analysis classes
  - Model analysis class interactions in Interaction diagrams



31

3.1. Allocating Responsibilities to Classes

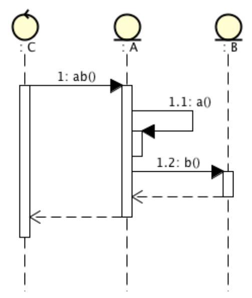
- Use analysis class stereotypes as a guide
  - Boundary Classes
    - Behavior that involves communication with an actor
  - Entity Classes
    - Behavior that involves the data encapsulated within the abstraction
  - Control Classes
    - Behavior specific to a use case or part of a very important flow of events

32

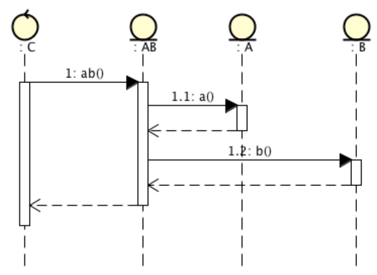
Responsibilities for the Entity classes

- (0) If one class has the data, put the responsibility with the data
- If multiple classes have the data:
  - (1) Put the responsibility with one class and add a relationship to the other
  - (2) Create a new class, put the responsibility in the new class, and add relationships to classes needed to perform the responsibility
  - (3) Put the responsibility in the control class, and add relationships to classes needed to perform the responsibility

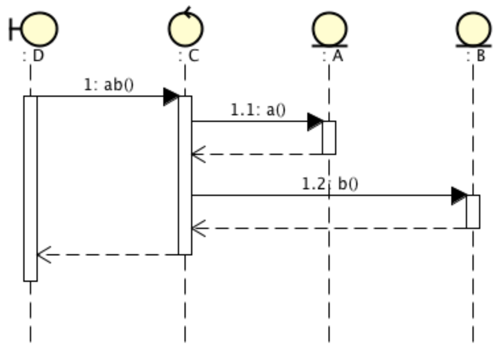
(1)



(2)



(3)

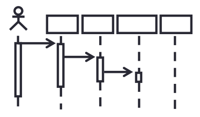


3.2. Interaction Diagrams

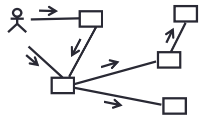
- Generic term that applies to several diagrams that emphasize object interactions
  - Sequence Diagram
  - Communication Diagram
- Specialized Variants
  - Timing Diagram
  - Interaction Overview Diagram

3.2. Interaction Diagrams (2)

- ◆Sequence Diagram
  - Time oriented view of object interaction


Sequence Diagrams

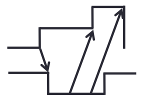
- ◆Communication Diagram
  - Structural view of messaging objects


Communication Diagrams

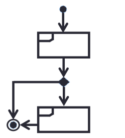
37

3.2. Interaction Diagrams (3)

- Timing Diagram
  - Time constraint view of messages involved in an interaction


Timing Diagrams

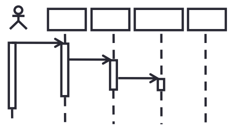
- Interaction Overview Diagram
  - High level view of interaction sets combined into logic sequence


Interaction Overview Diagrams

38

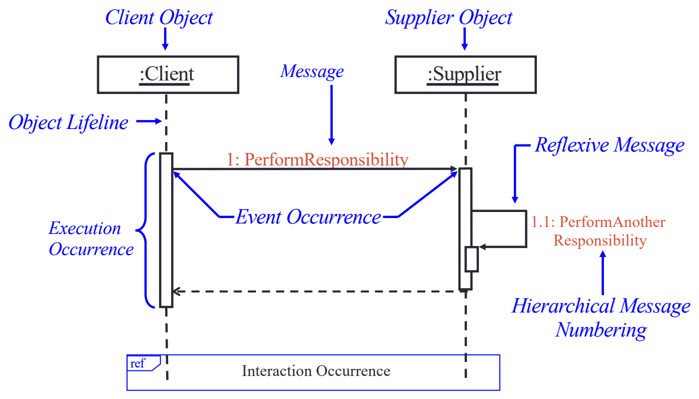
3.2.1. Sequence Diagram

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.
- The diagram shows:
  - The objects participating in the interaction.
  - The sequence of messages exchanged.


Sequence Diagram

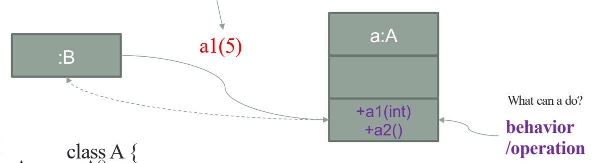
39

The Anatomy of Sequence Diagrams



40

message



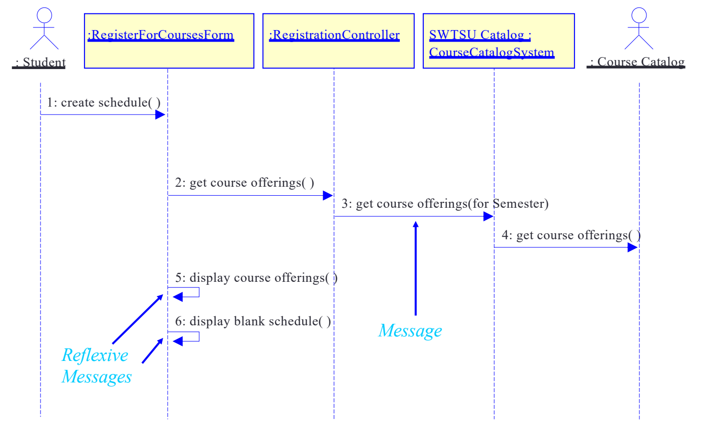
```
class B {
    A a = new A();

  +b(){
        a.a1(5);
  }
}
```

```
class A {
        - attr1;
        - attr2;
        - attr3;
        - attr4;

        +a1(int i){
                        //tính tiền trong ví
                        //trả về tổng số tiền
        }
}
```

method
= implementation of operation/behavior

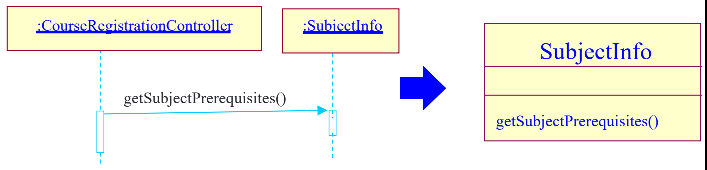
How can a do?

Sequence Diagram Contents: Messages


: Student
:RegisterForCoursesForm
:RegistrationController
SWTSU Catalog : CourseCatalogSystem
: Course Catalog
1: create schedule( )
2: get course offerings( )
3: get course offerings(for Semester)
4: get course offerings( )
5: display course offerings( )
6: display blank schedule( )
Message
Reflexive Messages

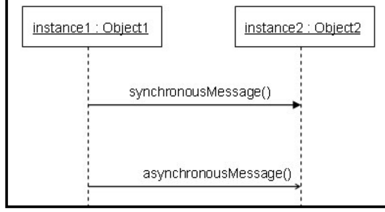
Messages in interation diagrams

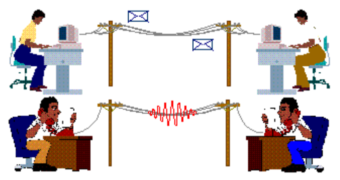
- Operation vs. Method?
- Call function vs. Send message
- Why sending messages?


:CourseRegistrationController
:SubjectInfo
getSubjectPrerequisites()
SubjectInfo
getSubjectPrerequisites()

Synchronized vs. asynchronized messages

- Synchronous message
  - Caller must wait until the message is done (e.g. invoking a subroutine)
- Asynchronous message
  - Caller can continue processing and doesn't have to wait for a response
  - Better responsiveness and reduces the temporal coupling but is harder to debug.
  - E.g. in multithreaded applications and in message-oriented middleware.





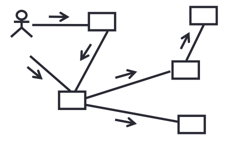
Exercise: AIMS

- Draw a sequence diagram for “Place order” use case

45

3.2.2. Communication Diagram

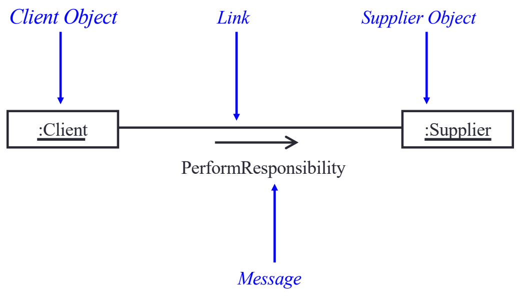
- A communication diagram emphasizes the organization of the objects that participate in an interaction.
- The communication diagram shows:
  - The objects participating in the interaction.
  - Links between the objects.
  - Messages passed between the objects.



Communication Diagrams

46

The Anatomy of Communication Diagrams



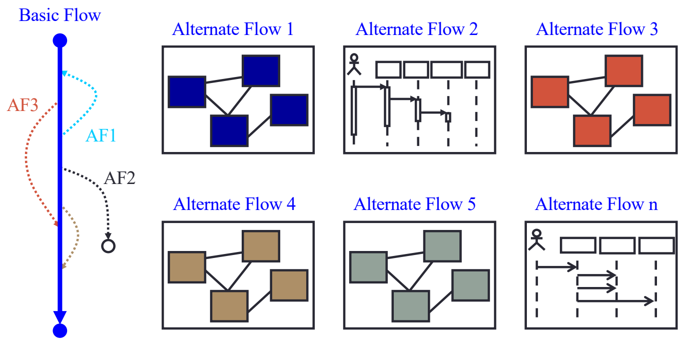
47

Exercise: AIMS

- Draw a communication diagram for “Place order” use case

48

One Interaction Diagram May Be Not Good Enough



49

3.2.3. Sequence and Communication Diagram Comparison

- Similarities
  - Semantically equivalent
    - Can convert one diagram to the other without losing any information
  - Model the dynamic aspects of a system
  - Model a use-case scenario

50

3.2.3. Sequence and Communication Diagram Comparison (2)

Sequence diagrams	Communication diagrams
Show the explicit sequence of messages	Show relationships in addition to interactions
Show execution occurrence	Better for visualizing patterns of communication
Better for visualizing overall flow	Better for visualizing all of the effects on a given object
Better for real-time specifications and for complex scenarios	Easier to use for brainstorming sessions

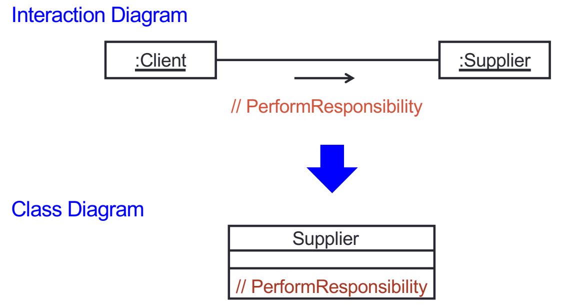
51

Content

52

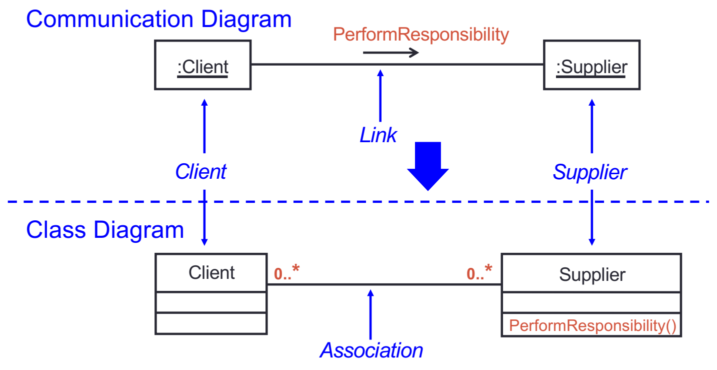
Describe Responsibilities

- What are responsibilities?
- How do I find them?



53

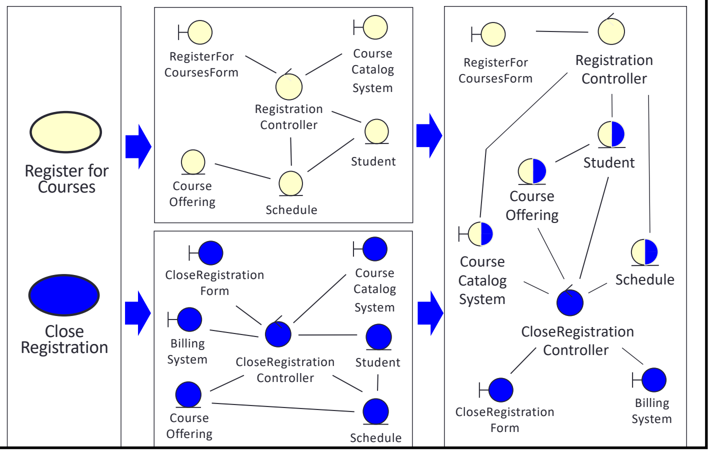
Finding Relationships



Relationship for every link!

54

Unify Analysis Classes

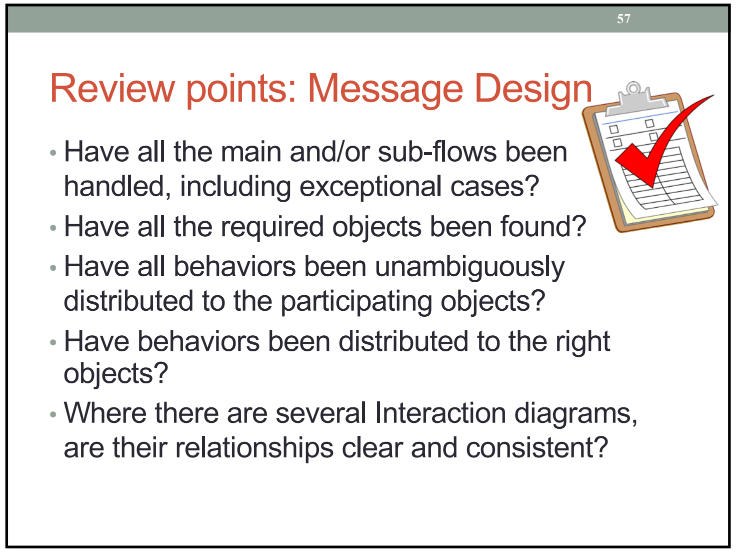


55

Reviewpoints: Analysis Classes

- Are the classes reasonable?
- Does the name of each class clearly reflect the role it plays?
- Does the class represent a single well-defined abstraction?
- Are all responsibilities functionally coupled?
- Does the class offer the required behavior?
- Are all specific requirements on the class addressed?

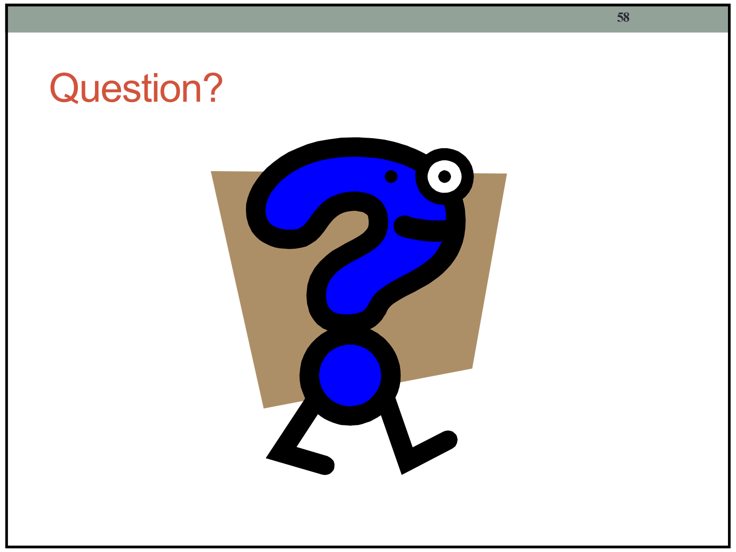
56



Review points: Message Design

- Have all the main and/or sub-flows been handled, including exceptional cases?
- Have all the required objects been found?
- Have all behaviors been unambiguously distributed to the participating objects?
- Have behaviors been distributed to the right objects?
- Where there are several Interaction diagrams, are their relationships clear and consistent?

57



Question?

58