Essentials of Systems Analysis and Design
Sixth Edition

Joseph S. Valacich
Joey F. George
Jeffrey A. Hoffer

Appendix A
Object-Oriented Analysis and Design

Learning Objectives

- Key terms
  - Association
  - Class diagram
  - Event
  - Object
  - Object class
  - Operation
  - Sequence diagram
  - State
  - State transition
  - Unified Modeling Language (UML)
  - Use case

Learning Objectives (continued)

- ✓ Discuss the concepts and principles underlying the object-oriented approach
- ✓ Learn to develop requirements models using use-case diagrams
- ✓ Learn to use class diagrams to develop object models of the problem domain
- ✓ Learn to develop requirements models using state and sequence diagrams

The Object-Oriented Modeling Approach

- Benefits
  1. The ability to tackle more **challenging** problem domains
  2. Improved **communication** among users, analysts, designers, and programmers
  3. **Reusability** of analysis, design, and programming results
  4. Increased **consistency** among the models developed during object-oriented analysis, design, and programming

The Object-Oriented Modeling Approach (continued)

- Object-Oriented Systems Development Life Cycle
  - Process of progressively developing representation of a system component (or object) through the phases of analysis, design, and implementation
  - The model is abstract in the early stages
  - As the model evolves, it becomes more and more detailed

The Object-Oriented Systems Development Life Cycle

- Analysis Phase
  - Model of the real-world application is developed showing its important properties
  - Model specifies the functional behavior of the system independent of implementation details
- Design Phase
  - Analysis model is refined and adapted to the environment
- Implementation Phase
  - Design is implemented using a programming language or database management system

The Object-Oriented Systems Development Life Cycle (continued)

- Unified Modeling Language (UML)
  - A notation that allows the modeler to specify, **visualize** and construct the artifacts of **software** systems, as well as **business models**
  - Techniques and notations
    - Use cases
    - **Class** diagrams
    - **State** diagrams
    - **Sequence** diagrams

Use-Case Modeling

- Use Case is a complete **sequence** of related actions initiated by an actor. Actor is an external entity that interacts with the system
- Use-Cases typically analyze **functional requirements** of the system
- Performed during the **analysis phase** to help developers understand functional requirements of the system **without regard for implementation details**

Use-Case Modeling

- Use cases represent functionality of the system
- Use cases may participate in relationships with other use cases
- Use cases may also use other use cases

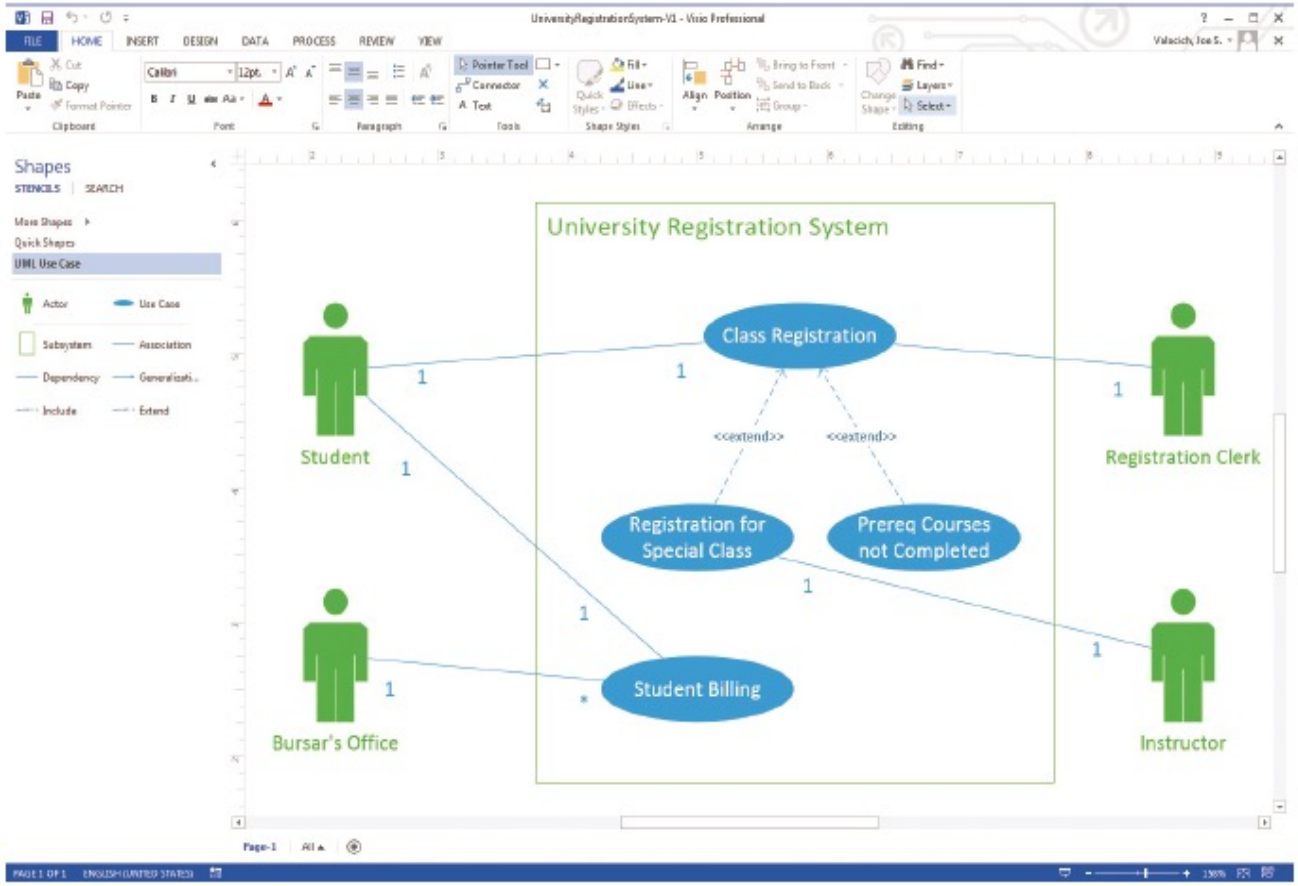


FIGURE A-1
Use-case diagram for a university registration system drawn using Microsoft Visio.

Object Modeling: Class Diagrams

- Object
  - An entity that has a well-defined role in the application domain, and has state, behavior, and identity
- State
  - A condition that encompasses an object's properties and the values those properties have
- Behavior
  - A manner that represents how an object acts and reacts
- Object Class
  - A set of objects that share a common structure and a common behavior

Object Modeling: Class Diagrams (continued)

- Class Diagram
  - **Class is represented as a rectangle** with three compartments
  - Objects can participate in relationships with objects of the same class

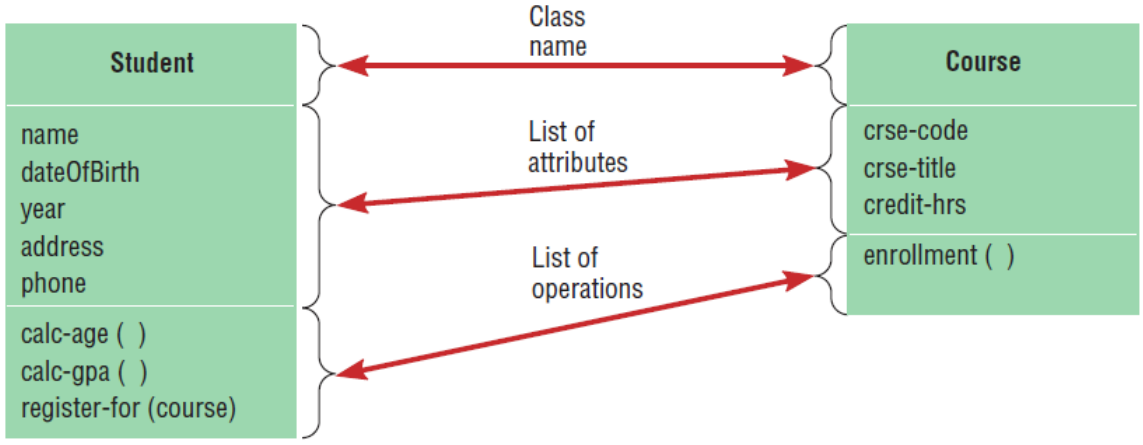


FIGURE A-3
UML class and object diagrams (A) Class diagram showing two classes (B) Object diagram with two instances.

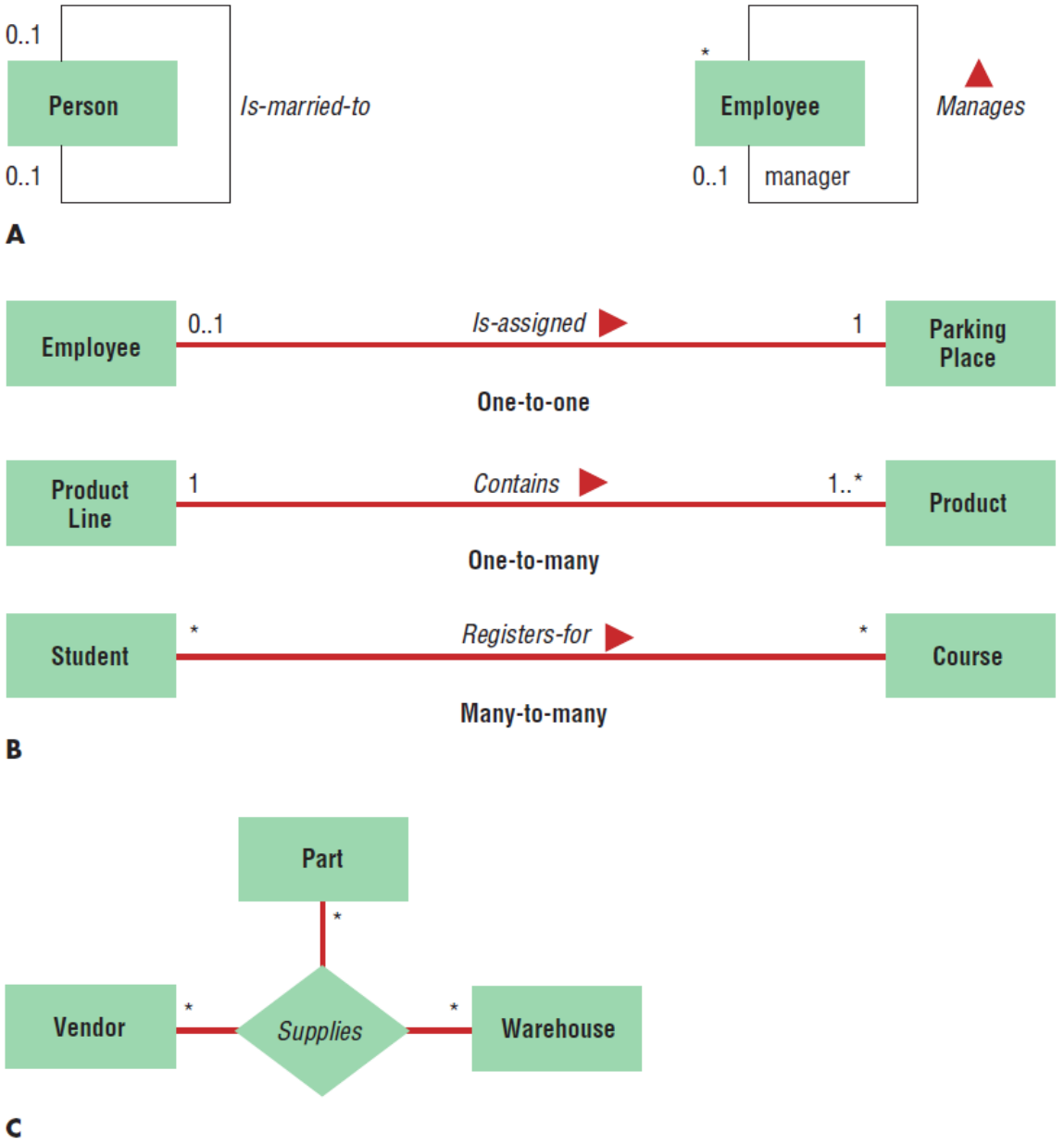
Object Modeling: Object Diagrams

- **Object** Diagram
  - A graph of **instances** that are **compatible with a given class diagram**; also called an instance diagram
  - Object is represented as a rectangle with two compartments
- Operation
  - A function or service that is provided by all the instances of a class
- Encapsulation
  - The technique of **hiding the internal implementation** details of an object from its external view

Representing Associations

- Association
  - A relationship between object classes
  - Degree may be unary, binary, ternary or higher
  - Depicted as a solid line between participating classes
- Association Role
  - The end of an association where it connects to a class
  - Each role has multiplicity, which indicates how many objects participate in a given association relationship

FIGURE A-4
Examples of association relationships of different degrees (A) Unary (B) Binary (C) Ternary.



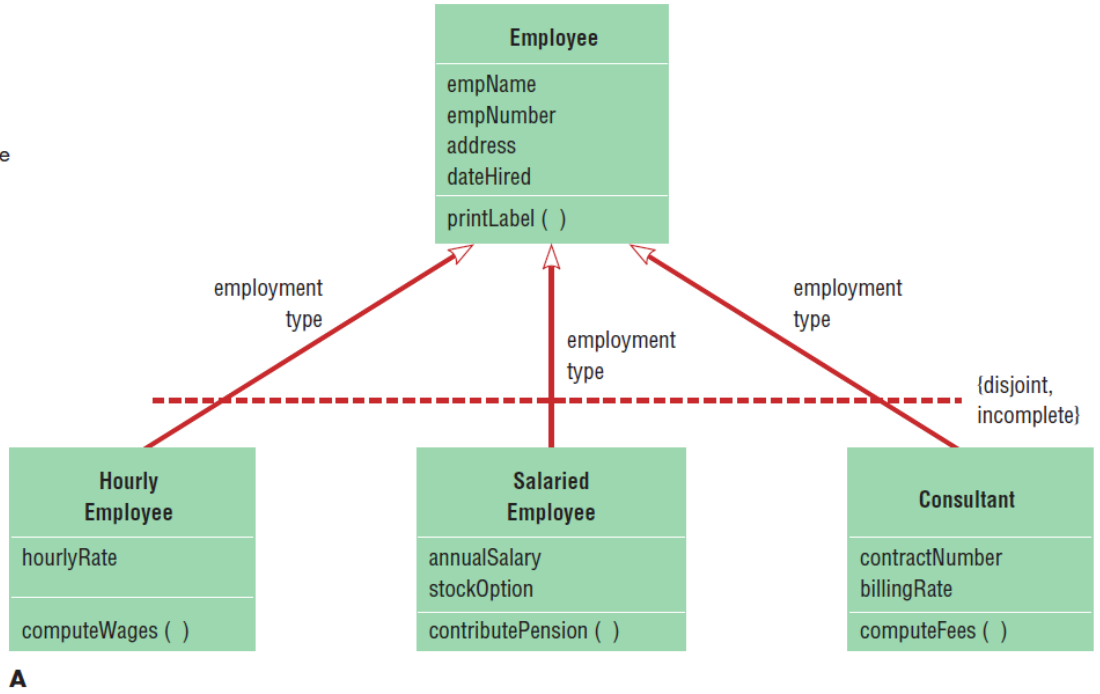
Representing Generalization

- Generalization
  - Abstraction of common features among multiple classes, as well as their relationships, into a more general class
- Subclass
  - A class that has been generalized
- Superclass
  - A class that is composed of several generalized subclasses

Representing Generalization (continued)

- Discriminator
  - Shows which property of an object class is being abstracted by a generalization relationship
- Inheritance
  - A property in which a subclass inherits the features from its superclass
- Abstract Class
  - A class that has no direct instances but whose descendents may have direct instances
- Concrete Class
  - A class that can have direct instances

FIGURE A-6
Examples of generalization, inheritance, and constraints (A) Employee superclass with three subclasses (B) Abstract patient class with two concrete subclasses.



Representing Aggregation

- Aggregation
  - An **aggregation** expresses a part-of relationship between a component object and an aggregate object. It is a stronger form of association relationship (with the added "part-of" semantics) and is represented with a hollow diamond at the aggregate end.
  - "Composed of" relationship. A part-of relationship between a component object and an aggregate object
  - A solid diamond represents a stronger form of aggregation known as **composition**. In composition, a part object **belongs** to **only one** whole object; for example, a room is part of only one building. Therefore, the multiplicity on the aggregate end may not exceed one.
  - Example: Personal computer
    - Composed of CPU, Monitor, Keyboard, etc.

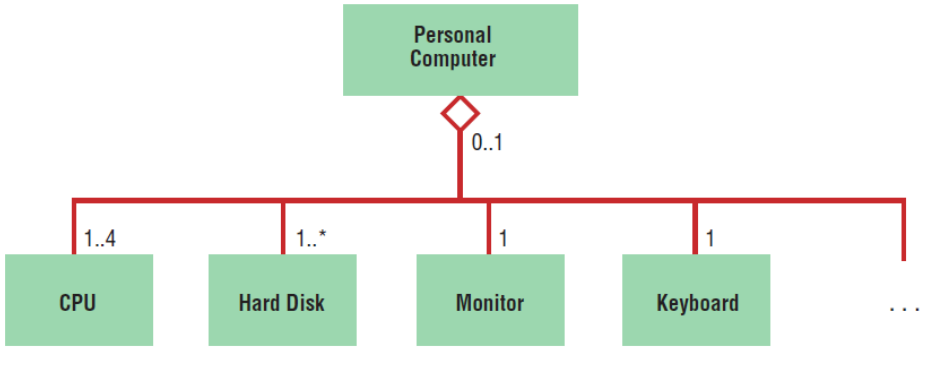
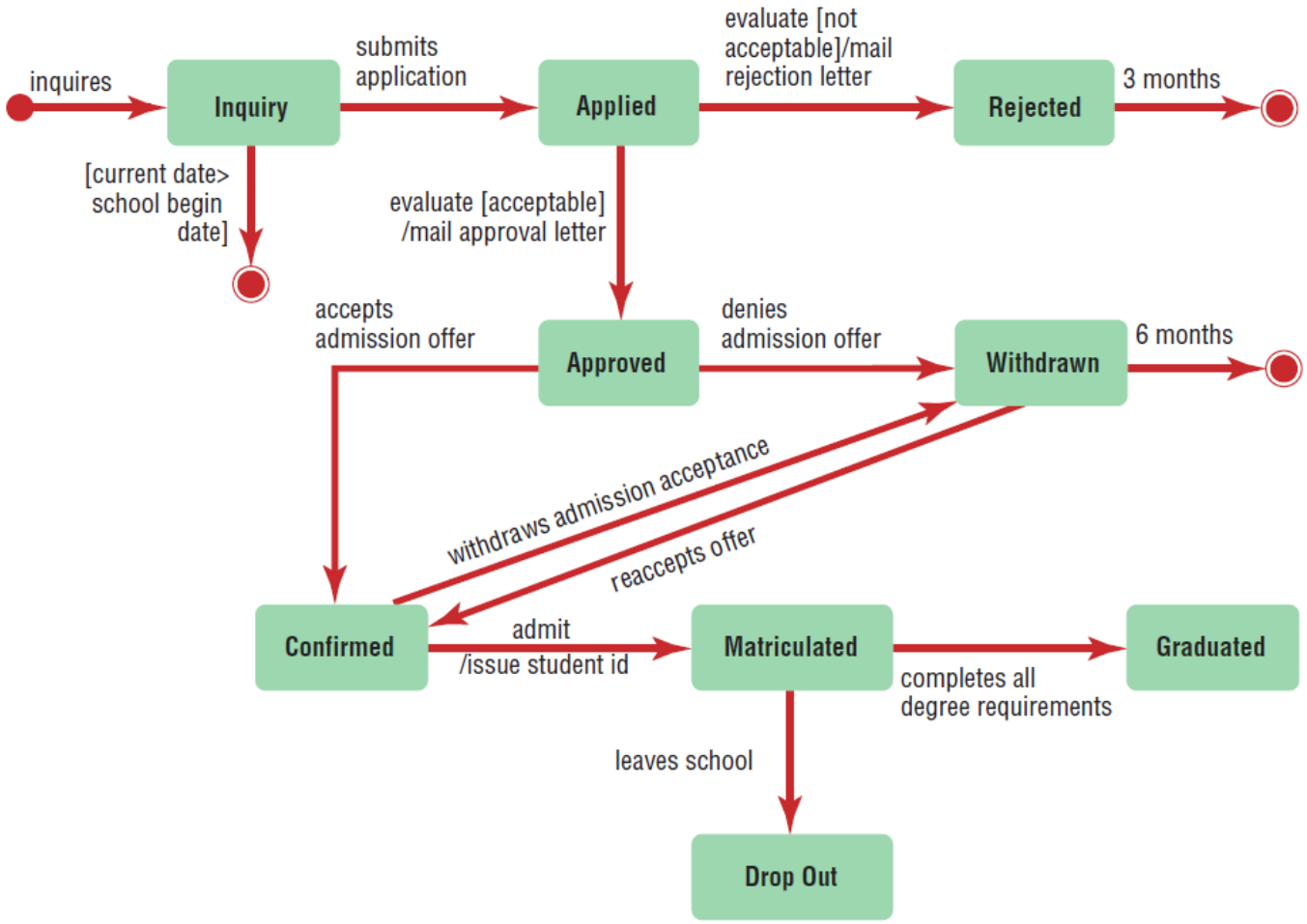


FIGURE A-7
Example of aggregation.

Dynamic Modeling: State Diagrams

- State
  - A condition during the life of an object during which it satisfies some conditions, performs some actions or waits for some events
  - Shown as a rectangle with rounded corners
- State Transition
  - The changes in the attributes of an object or in the links an object has with other objects
  - Shown as a solid arrow
  - Diagrammed with a guard condition and action
- Event
  - Something that takes place at a certain point in time

FIGURE A-8
State diagram for the Student object.



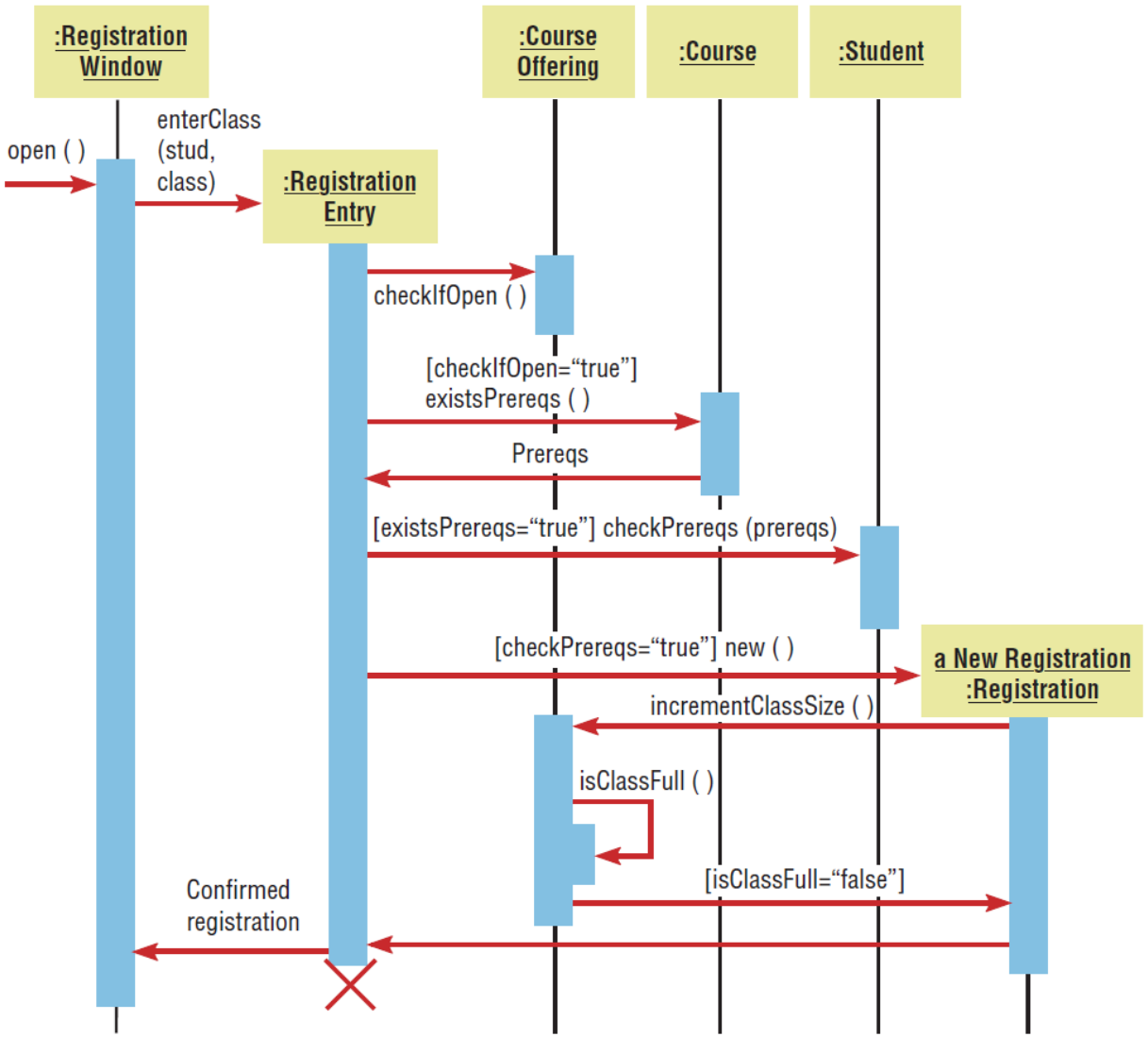
Dynamic Modeling: Sequence Diagrams

- Sequence Diagram
  - A depiction of the interaction among objects during certain periods of time
- Activation
  - The time period during which an object performs an operation
- Messages
  - Means by which objects communicate with each other

Dynamic Modeling: Sequence Diagrams (continued)

- Synchronous Message
  - A type of message in which the caller has to wait for the receiving object to finish executing the called operation before it can resume execution itself
- Simple Message
  - A message that transfers control from the sender to the recipient without describing the details of the communication

FIGURE A-9
Sequence diagram for a class registration scenario with prerequisites.



- Open() is a Simple Message

Moving to Design

- Start with existing set of analysis model
- Progressively add technical details
- Design model must be more detailed than analysis model
- Component Diagram
  - A diagram that shows the software components or modules and their dependencies
- Deployment Diagram
  - A diagram that shows how the software components, processes and objects are deployed into the physical architecture of the system

Component Diagrams

- Component Diagrams are a type of structural diagram that shows the organization and dependencies among a set of components. Components represent modular parts of a system that encapsulate its contents and whose implementation and deployment can be done independently. These diagrams are particularly useful for visualizing, specifying, and documenting component-based systems and for constructing executable systems through forward and reverse engineering.
- Rectangles: The primary elements in a component diagram are represented as rectangles. These rectangles depict components, which are modular parts of a system. A component is typically illustrated with a name and may also include internal details such as operations or interfaces. Components are self-contained and are designed to be reused, replaced, or updated without affecting the rest of the system.
- Circles represent the provided interfaces of a component. These interfaces indicate the services that the component offers to its environment. A lollipop interface is a shorthand notation for a realization relationship from a component to an interface.
- Sockets: The counterpart to the lollipop symbol, often represented by a half-circle or a socket shape, indicates required interfaces—these detail the services a component expects from its environment.

Component Diagrams

- Solid lines are often used to represent relationships between components and interfaces. Specifically, a solid line with a closed arrowhead represents a dependency relationship. This implies that one component depends on another component or an interface for its functionality.
- Dashed lines are used to denote different types of relationships. A common usage is for representing realization relationships, particularly between a component and the interfaces it provides (illustrated with lollipops). This denotes that the component implements the functionalities defined in the interface. Dashed lines can also indicate usage or dependency relationships where the specifics of the relationship are defined by the nature of the connection between the elements (e.g., a component requiring an interface).

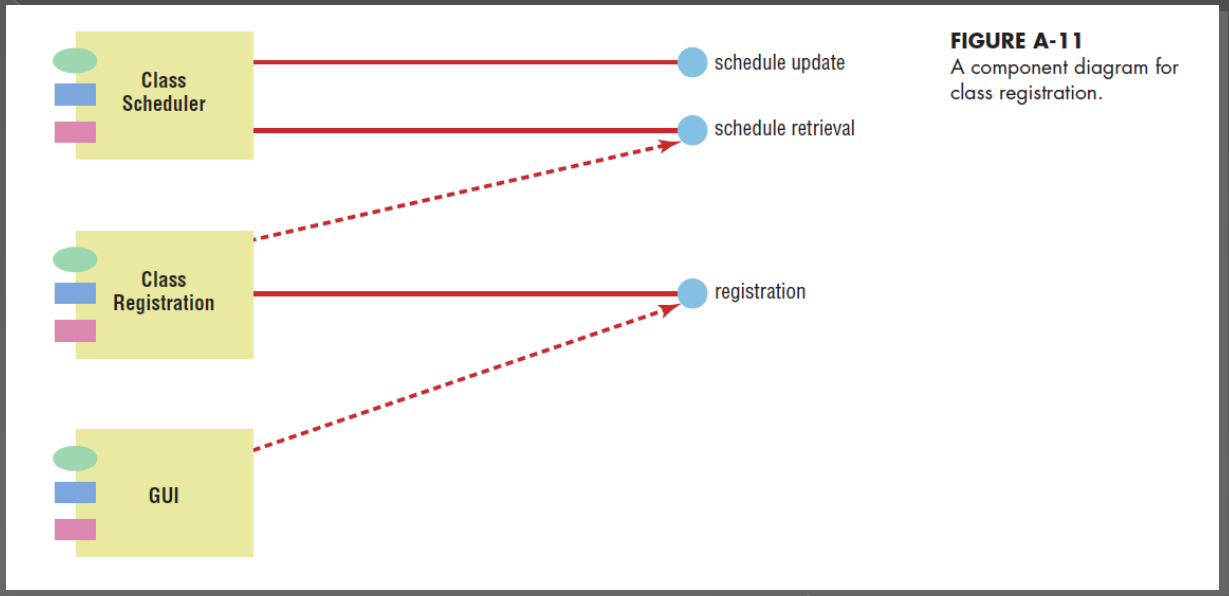


FIGURE A-11
A component diagram for class registration.

Three software components we have are: Class Scheduler, Class Registration, and GUI. The small circles in the diagram represent interfaces. The registration interface, for example, is used to register a student for a class, and the schedule update interface is used for updating a class schedule.

Summary

- Object-Oriented Modeling Approach
  - Benefits
  - Unified Modeling Language
    - Use cases
    - Class diagrams
    - State diagrams
    - Sequence diagrams
- Use Case Modeling

Summary (continued)

- Object Modeling: Class Diagrams
  - Associations
  - Generalizations
  - Aggregation, Composition
- Dynamic Modeling: State Diagrams
- Dynamic Modeling: Sequence Diagrams
- Moving to Design