دانشگاه کردستان
University of Kurdistan
زانکۆی کوردستان

Overview of Software Architecture

Sadegh Sulaimany
info@Bioinfotmation.ir

Software Architecture Course

Initial assessment

1. What is software architectural patterns?
   List at least 3 of them that are familiar for you.
   Just write them and do not explain

2. What is asynchronous message in sequence diagram and what about its difference with normal message communication in case of drawing?

General Definition

› Software architecture
  – separates the overall structure of the system,
  in terms of subsystems and their interfaces,
  from the internal details of the individual subsystems

› A software architecture
  – is **structured into subsystems,**
    in which each subsystem should be relatively independent of other subsystems

  – is high level design

Definition

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."

- Thus
  - A software architecture is considered primarily from a structural perspective
  - However
    - In order to fully understand the software architecture,
    - it is also necessary to study it from several perspectives,
      - including both static and dynamic perspectives

Structural perspective of SA

- Component-Based Software Architecture
  - consists of multiple components in which each component is self-contained and encapsulates certain information
  - A component is either a composite object composed of other objects or a simple object.
  - A component provides an interface through which it communicates with other components.
    - All information that is needed by one component to communicate with another component is contained in the interface, which is separate from the implementation.
- Thus,
  - a component can be considered a black box,
    - because its implementation is hidden from other components. Components communicate with each other in different ways using predefined communication patterns.

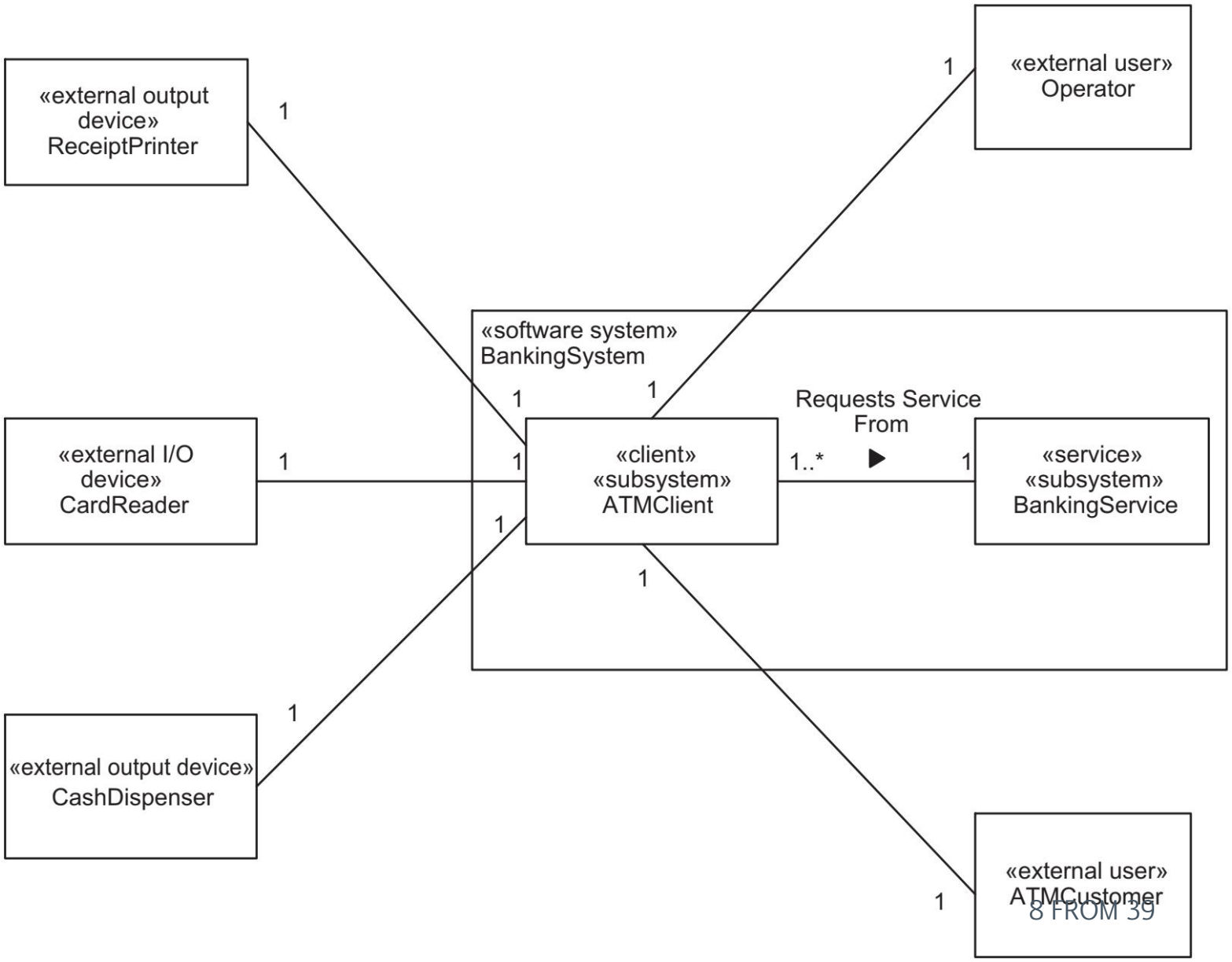
Software Architecture views

- Structural View (Module view)
- Dynamic View (Component & Connector view)
- Deployment View (Allocation view)

Software Architecture views

- Structural View
  - A static view, which does not change with time
  - At the highest level, subsystems are depicted on a class diagram
  - Example
    - Structural view of client/server software architecture: high-level class diagram for Banking System

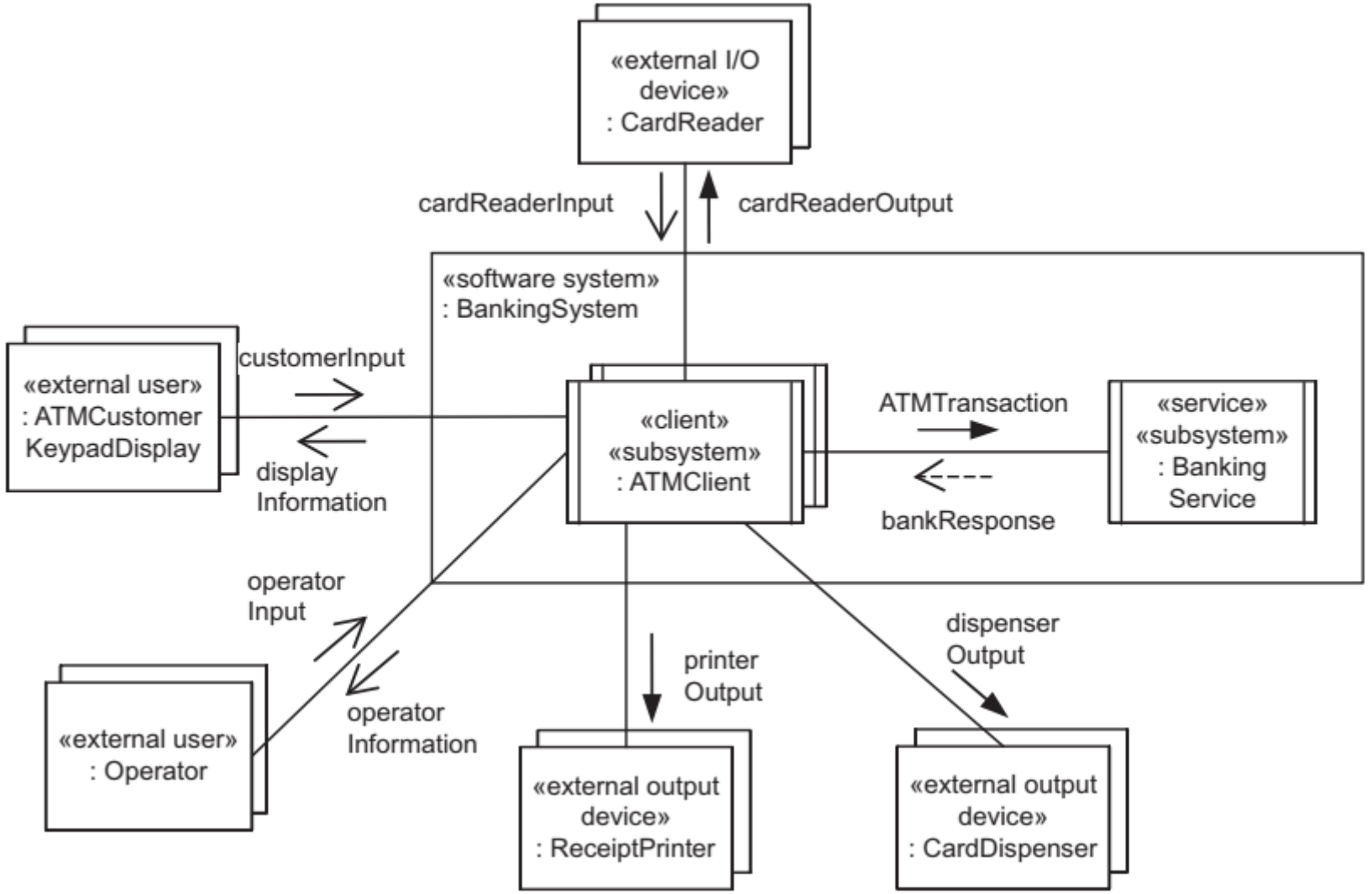
Structural View



Dynamic view

- Is a behavioral view, which is depicted on a communication diagram
- A subsystem communication diagram
  - shows the subsystems (depicted as aggregate or composite objects) and the message communication between them.
- Example
  - Dynamic view of client/server software architecture: high-level communication diagram for Banking System

Dynamic view

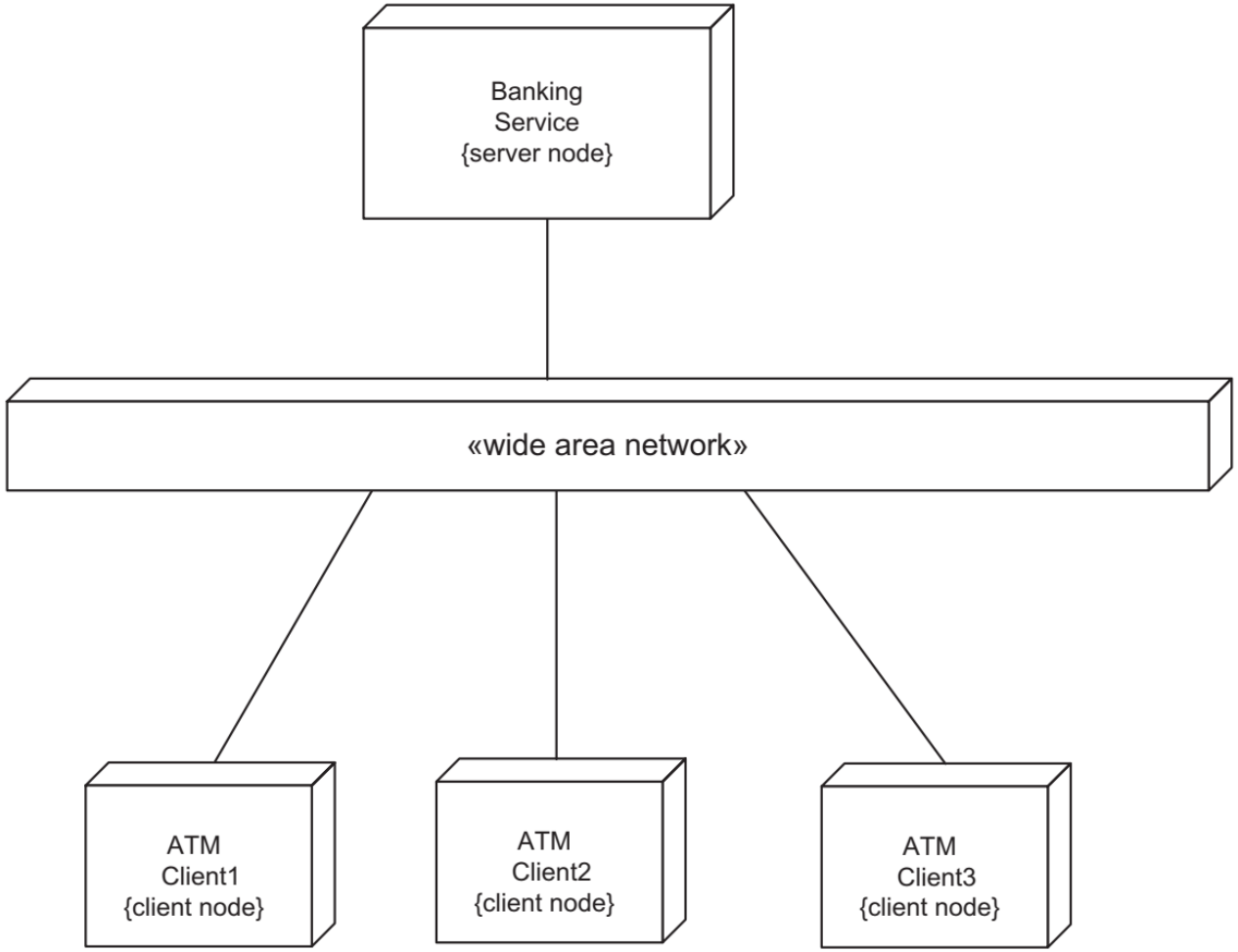


Deployment view

- depicts the physical configuration of the software architecture,
- in particular how the subsystems of the architecture are allocated to physical nodes
- can depict a specific deployment with a fixed number of nodes

– Example
  - Deployment view of client/server software architecture

Deployment view



Software architectural patterns

- provide the skeleton or template for the overall software architecture or high-level design of an application
  - referred to architectural styles or patterns of software architecture,
    - which are recurring architectures used in a variety of software applications
  - can be grouped into two main categories:
    1. architectural structure patterns, which address the static structure of the architecture
    2. architectural communication patterns, which address the dynamic communication among distributed components of t

Software architectural patterns

1. Architectural structure pattern

   - Centralized Control Pattern
   - Distributed Control Pattern
   - Hierarchical Control Pattern
   - Layers of Abstraction Pattern
   - Multiple Client/Multiple Service Pattern
   - Multiple Client/Single Service Pattern
   - Multi-tier Client/Service Pattern

Software architectural patterns

2. Architectural communication patterns
   - Asynchronous Message Communication Pattern
   - Asynchronous Message Communication with Callback Pattern
   - Bidirectional Asynchronous Message Communication
   - Broadcast Pattern
   - Broker Forwarding Pattern
   - Broker Handle Pattern
   - Call/Return
   - Subscription/Notification Pattern
   - Synchronous Message Communication with Reply Pattern
   - Synchronous Message Communication without Reply Pattern

- Negotiation Pattern
- Service Discovery Pattern
- Service Registration

Software architectural patterns

1. Architectural structure pattern

   A common one:

   – Layers of Abstraction pattern

   › also known as the Hierarchical Layers or Levels of Abstraction pattern

   › a common architectural pattern that is applied in many different software domains

   › Examples

   – Operating systems, database management systems, and network communication software

   › it can be extended by the addition of upper layers that use services provided by lower layers

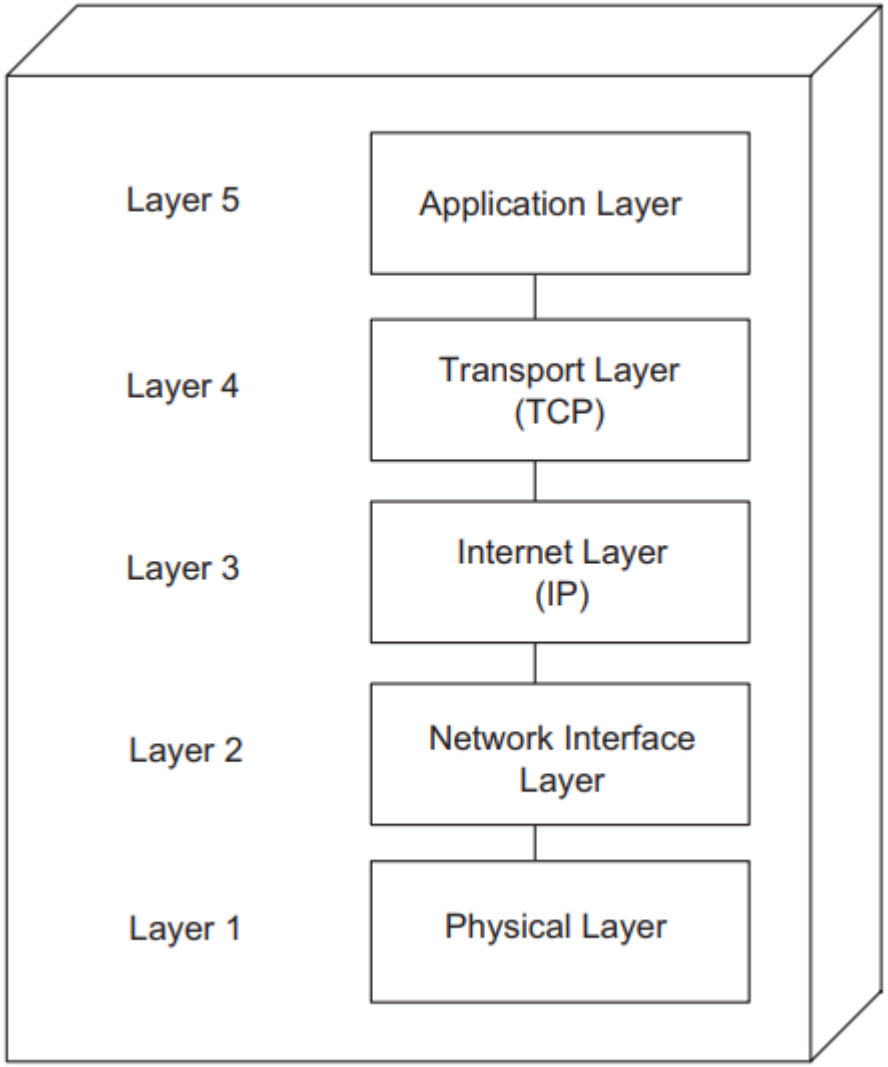
   and contracted by the removal of upper layers

   › each layer uses services in the layer immediately below it

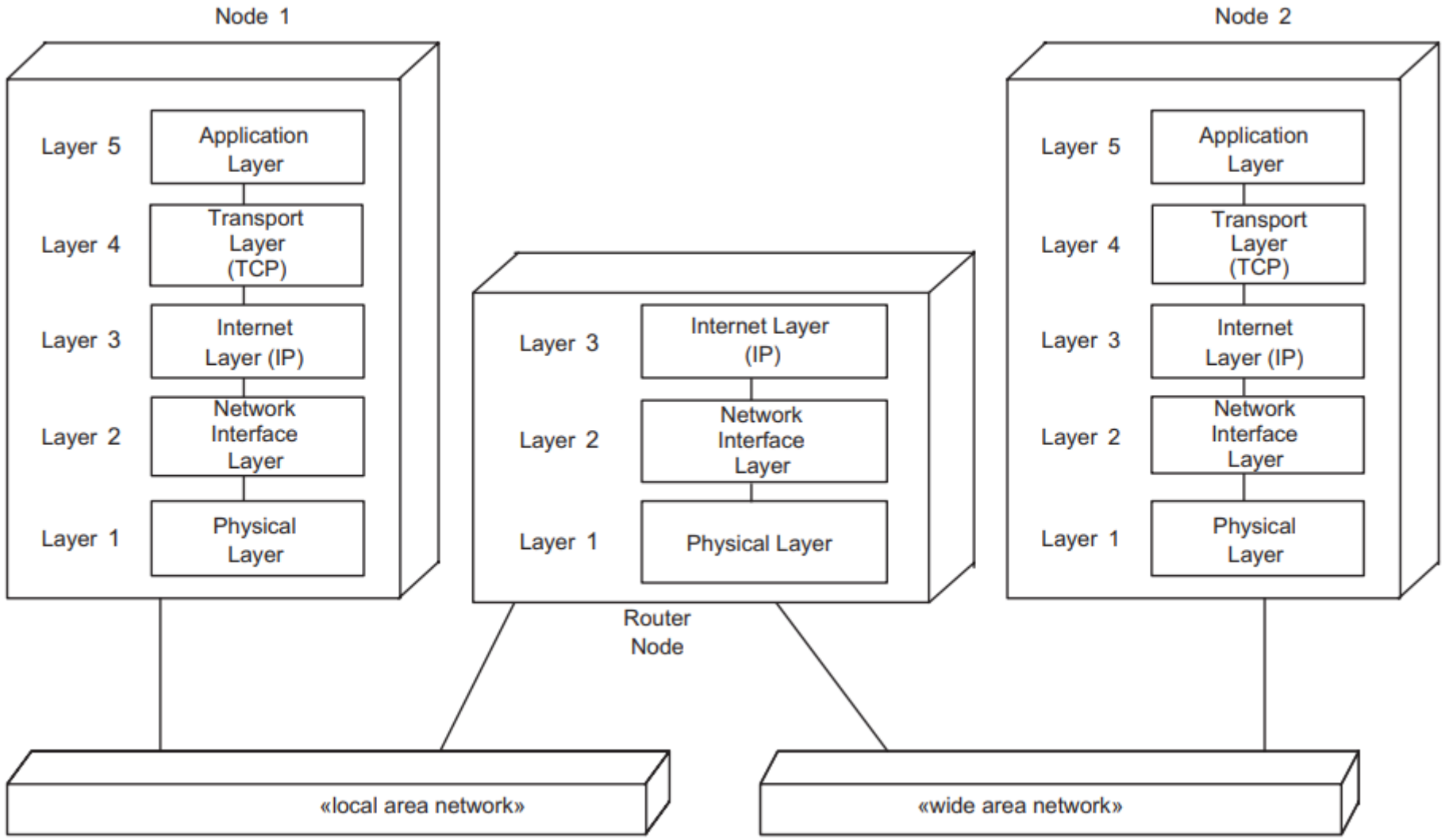
   – Or layer not immediately below it

Software architectural patterns

1. Architectural structure pattern
   - Layers of Abstraction pattern
   - Example
     - TCP/IP software layers of abstraction



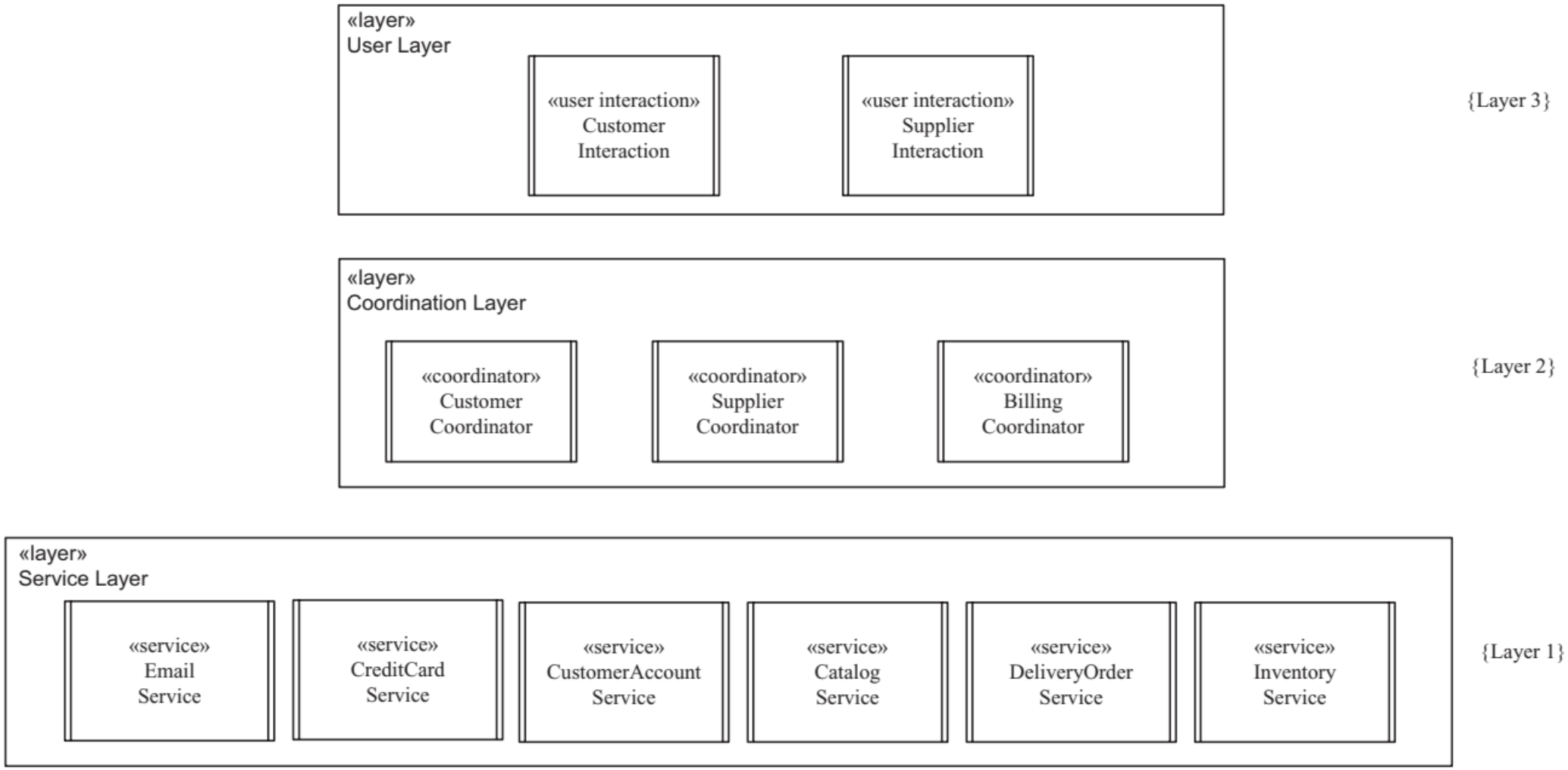
Software architectural patterns



ture
by the

l,

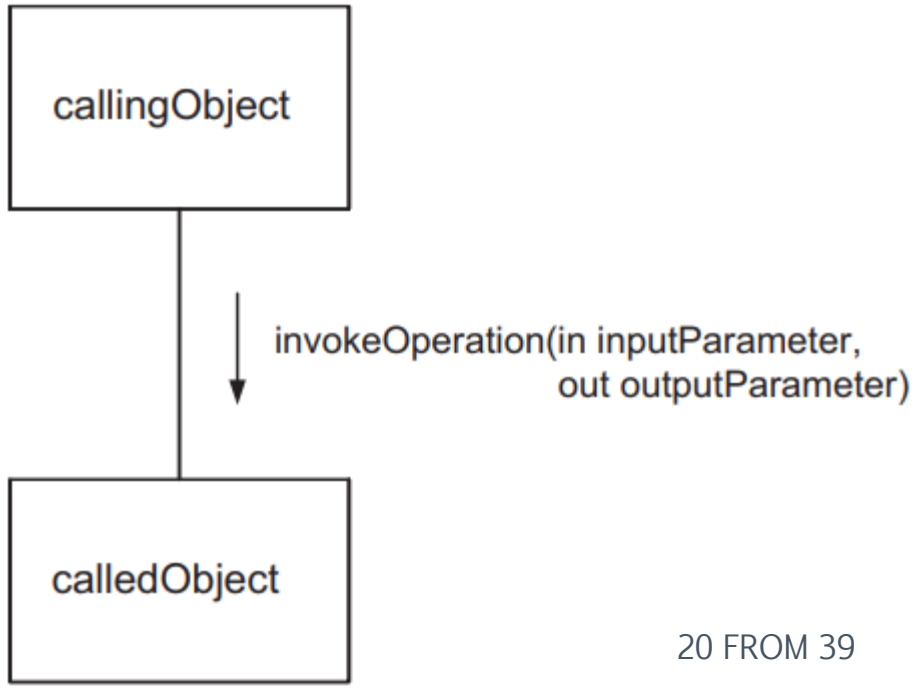
Software architectural patterns



Software architectural patterns

2. Architectural communication patterns

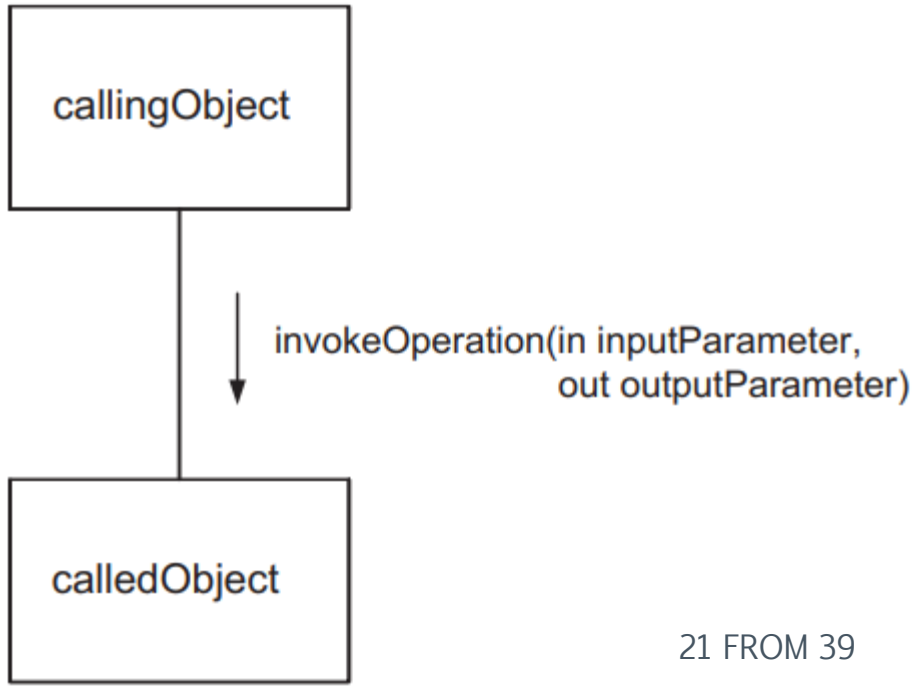
- Call/Return
  - simplest form of communication between objects uses the Call/Return pattern
  - In this pattern, a calling operation in the calling object invokes a called operation in the called object,
  - Change control form calling to callee
  - Parameters passed to callee
  - When called method finished control returns to calling object
  - synchronous communication
    - **arrow with black arrowhead**



Software architectural patterns

2. Architectural communication patterns

- Call/Return
  - simplest form of communication between objects uses the Call/Return pattern
  - In this pattern, a calling operation in the calling object invokes a called operation in the called object,
  - Change control form calling to callee
  - Parameters passed to callee
  - When called method finished control returns to calling object
  - synchronous communication
    - **arrow with black arrowhead**

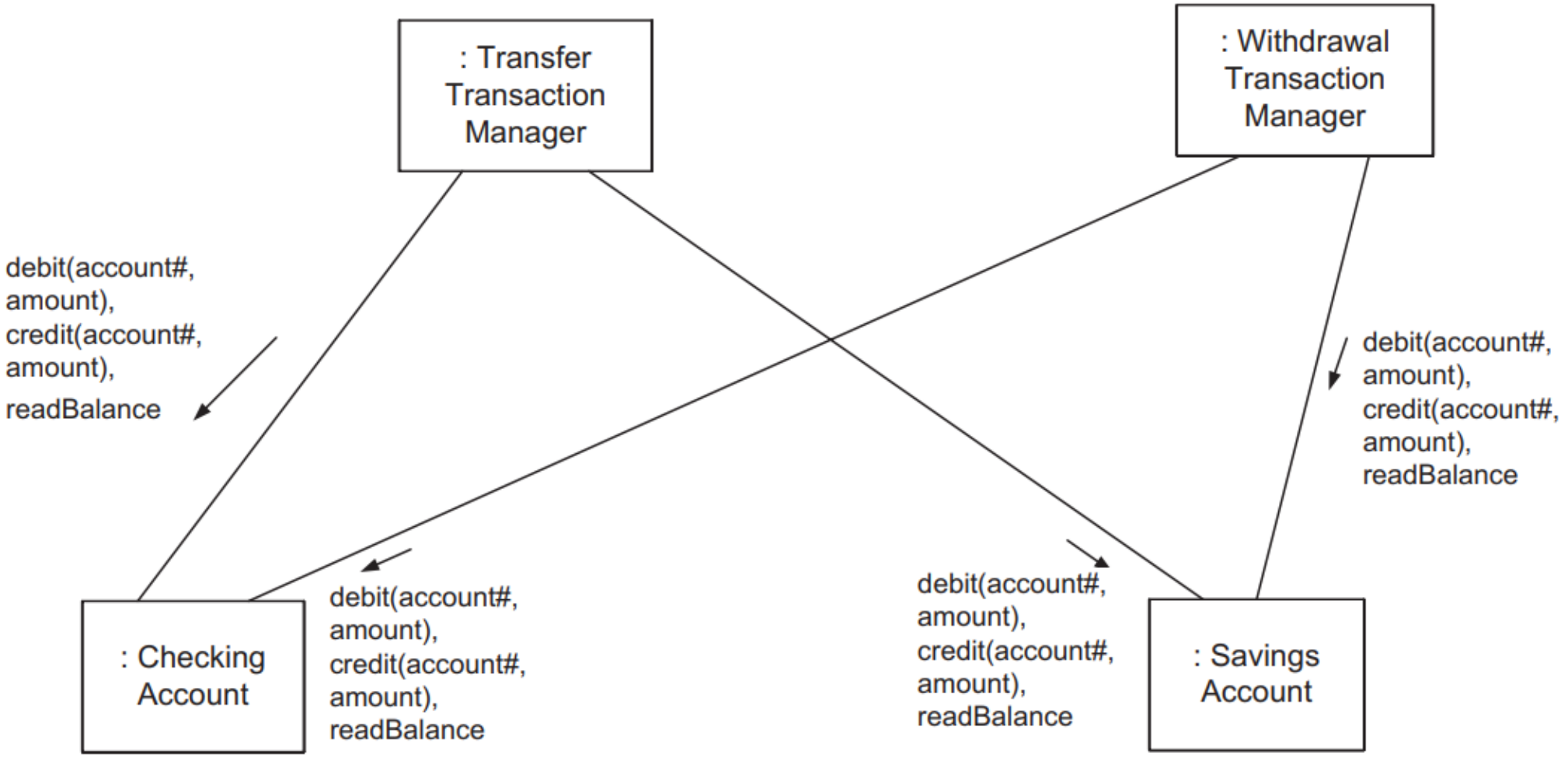


Software architectural patterns

2. Architectural communication patterns

- Call/Return
  - Example
    - a sequential design with instance of the checking account and savings account classes
    - Each object provides credit and debit operations
    - can be invoked by the Withdrawal Transaction Manager or Transfer Transaction Manager objects

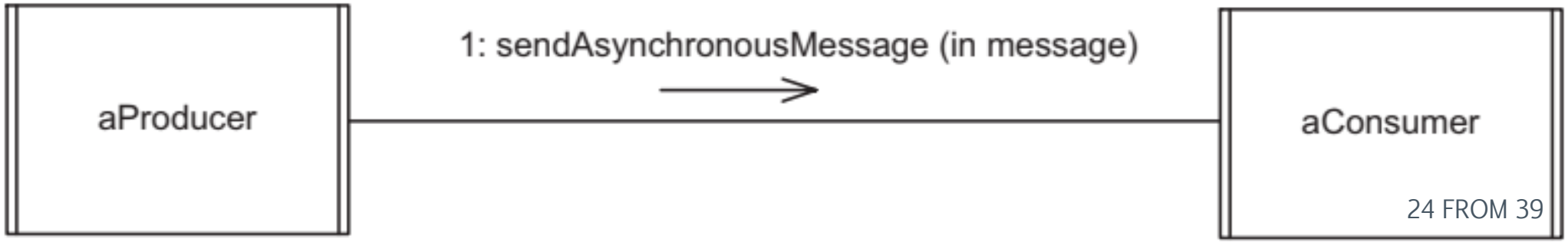
Software architectural patterns



Software architectural patterns

2. Architectural communication patterns

- Asynchronous Message Communication Pattern
  - For concurrent and distributed systems
  - Another name: Loosely Coupled Message Communication pattern
  - Main property
    - producer component sends a message to the consumer component and does not wait for a reply
    - The producer continues because it either does not need a response or has other functions to perform before receiving a response
    - The consumer receives the message; if the consumer is busy when the message arrives, the message is queued



Software architectural patterns

2. Architectural communication patterns

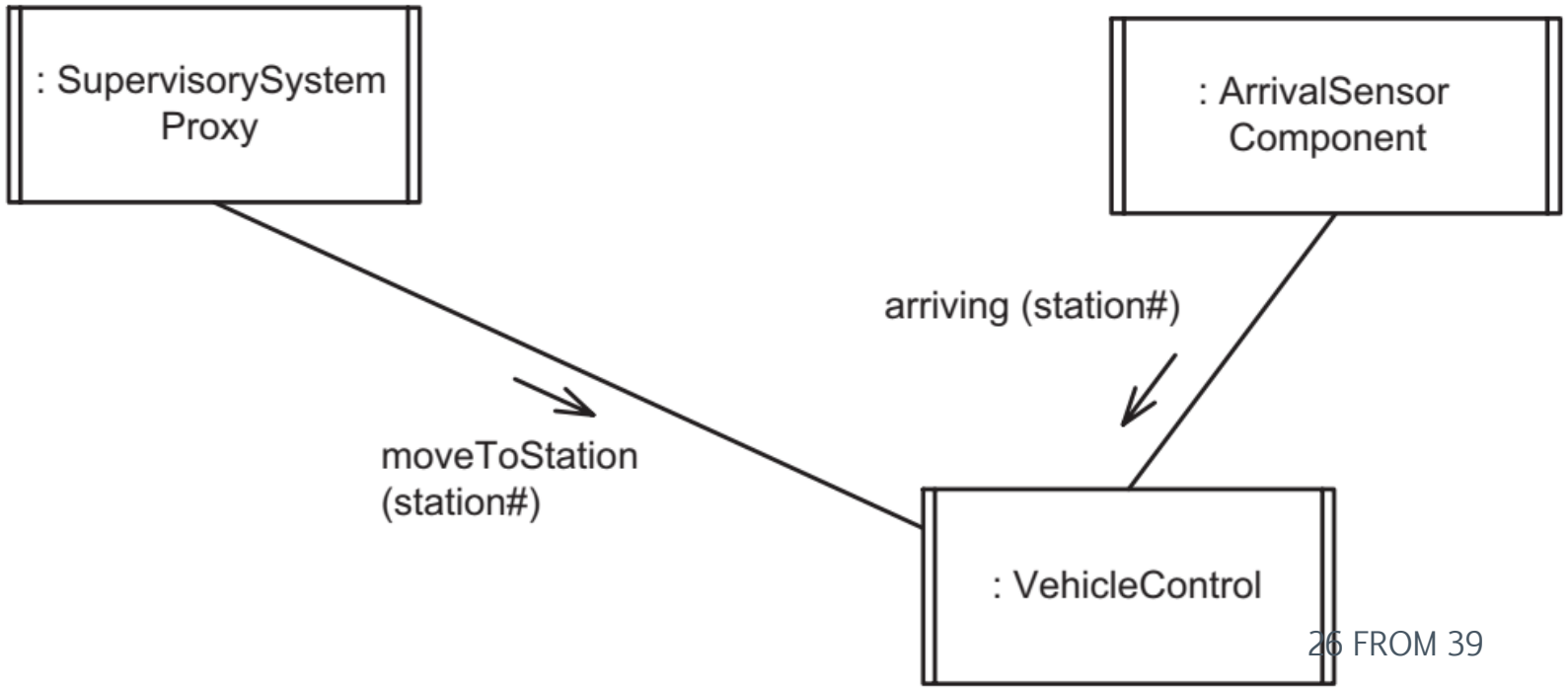
– Asynchronous Message Communication Pattern

- the producer and consumer components proceed asynchronously (i.e., at different speeds), a first-in, first out (FIFO) message queue can build up between producer and consumer
- If no message is available when the consumer requests one, the consumer is suspended until next message arrival
- Asynchronous Message Communication pattern is used wherever possible for greater flexibility.
- This approach can be used if the sender does not need a response from the receiver
- arrow with stick arrowhead = Asynchronous Message Communication

Software architectural patterns

2. Architectural communication patterns

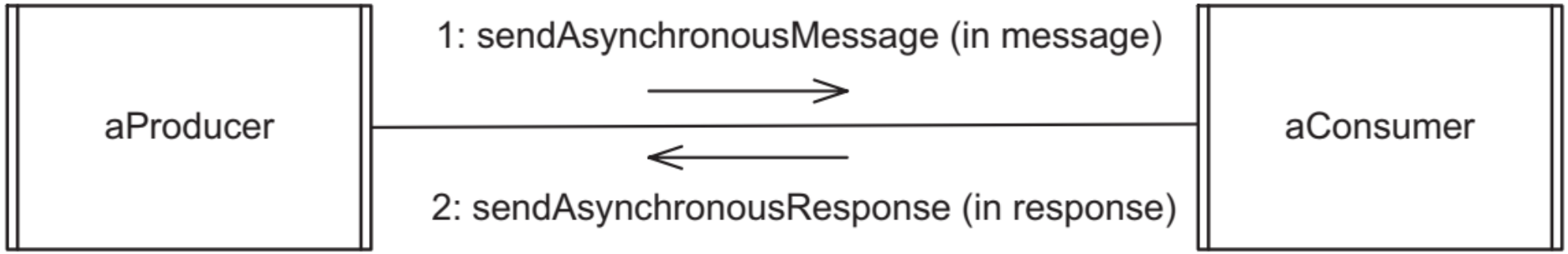
- Asynchronous Message Communication Pattern
  - Example
    - Automated Guided Vehicle System
    - in a distributed environment



Software architectural patterns

2. Architectural communication patterns

- Asynchronous Message Communication Pattern
  - In peer-to-peer communication
    - Two component may send asynchronous messages to each other
    - **bidirectional asynchronous communication**
    -



Software architectural patterns

2. Architectural communication patterns
   - Asynchronous Message Communication Pattern
   - Asynchronous Message Communication with Callback Pattern
   - Bidirectional Asynchronous Message Communication
   - Broadcast Pattern
   - Broker Forwarding Pattern
   - Broker Handle Pattern
   - Call/Return
   - Subscription/Notification Pattern
   - Synchronous Message Communication with Reply Pattern
   - Synchronous Message Communication without Reply Pattern

- Negotiation Pattern
- Service Discovery Pattern
- Service Registration

Software architectural patterns

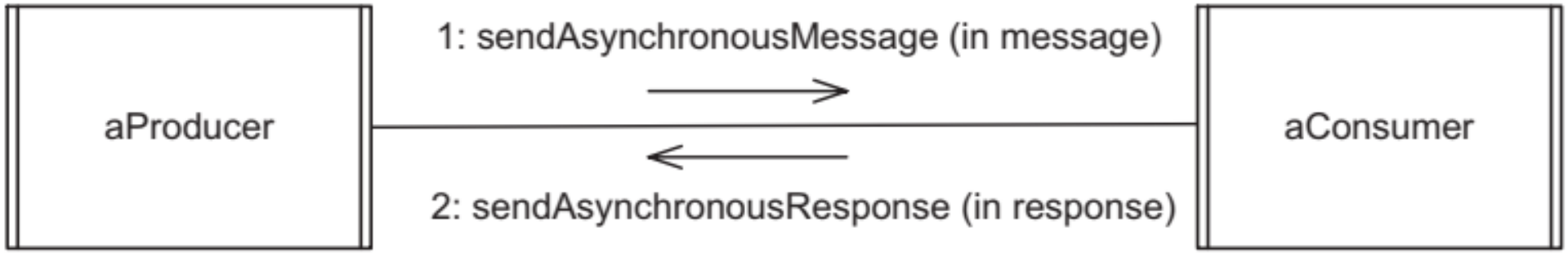
2. Architectural communication patterns

- Synchronous Message Communication Pattern with Reply Pattern
  - also referred to as Tightly Coupled Message Communication with Reply pattern
  - the client component sends a message to the service component and then waits for a reply from the service
  - When the message arrives, the service accepts it, processes it, generates a reply, and then sends the reply

Software architectural patterns

2. Architectural communication patterns

- Synchronous Message Communication Pattern with Reply Pattern
  - it is more likely that synchronous message communication involves multiple clients and one service
    - the is fundamental to client/server architectures
  - Bidirectional Asynchronous Message Communication pattern



Documenting Software Architectural Patterns

- it is very useful to have a standard way of describing and documenting a pattern
  - so that it can be easily referenced, compared with other patterns, and reused

- Three important aspects of a pattern that need to be captured are
  - Context
    - the situation that gives rise to a problem
  - Problem
    - refers to a recurring problem that arises in this context
  - Solution
    - a proven resolution to the problem

Documenting Software Architectural Patterns

- A template for describing a pattern usually also addresses its strengths, weaknesses, and related patterns.
  - A typical template looks like this:

    - **Pattern name**
    - **Aliases.** Other names by which this pattern is known.
    - **Context.** The situation that gives rise to this problem.
    - **Problem.** Brief description of the problem.
    - **Summary of solution.** Brief description of the solution.
    - **Strengths of solution**
    - **Weaknesses of solution**
    - **Applicability.** When you can use the pattern.
    - **Related patterns**
    - **Reference.** Where you can find more information about the pattern.

Documenting Software Architectural Patterns

- An example
  - of documenting a pattern is given next for the Layered Pattern

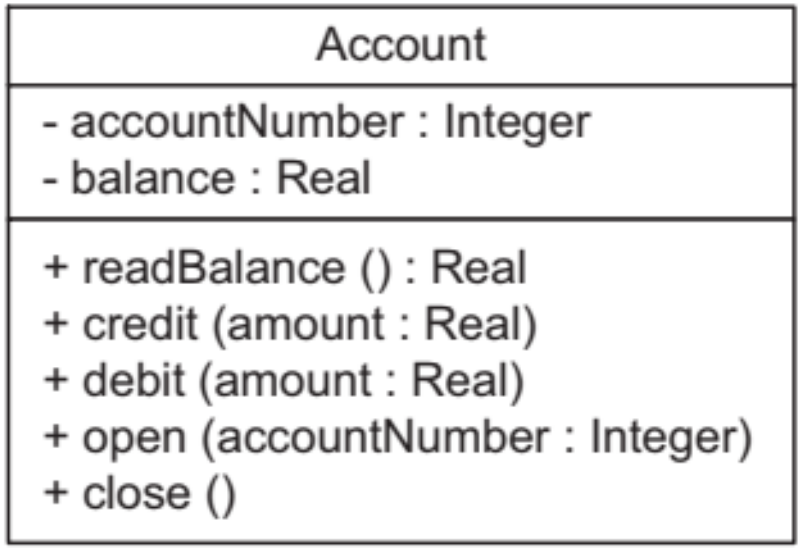
Pattern name	Layers of Abstraction
Aliases	Hierarchical Layers, Levels of Abstraction
Context	Software architectural design
Problem	A software architecture that encourages design for ease of extension and contraction is needed.
Summary of solution	Components at lower layers provide services for components at higher layers. Components may use only services provided by components at lower layers.
Strengths of solution	Promotes extension and contraction of software design
Weaknesses of solution	Could lead to inefficiency if too many layers need to be traversed
Applicability	Operating systems, communication protocols, software product lines
Related patterns	Kernel can be lowest layer of Layers of Abstraction architecture. Variations of this pattern include Flexible Layers of Abstraction.
Reference	Chapter 12, Section 12.3.1; Hoffman and Weiss 2001; Parnas 1979.

Interface design

- An important goal of
  - both object-oriented design and component-based software architecture
    - is the separation of the interface from the implementation
  - An **interface** specifies the externally visible operations of a class, service, or component without revealing the internal structure (implementation) of the operations
  - The interface can be considered a *contract* between the designer of the external view of the class and the implementer of the class internals.
  - It is also a *contract* between a class that requires (uses) the interface (i.e., invokes the operations provided by the interface) and the class that provides the interface.

Interface design

- Following the concept of information hiding
  - class attributes are private and the public operations provided by a class constitute the interface
  - In static modeling using class diagram notation,
    - *the interface (class operations) is depicted in the third compartment of the class* (+ vs - sign)



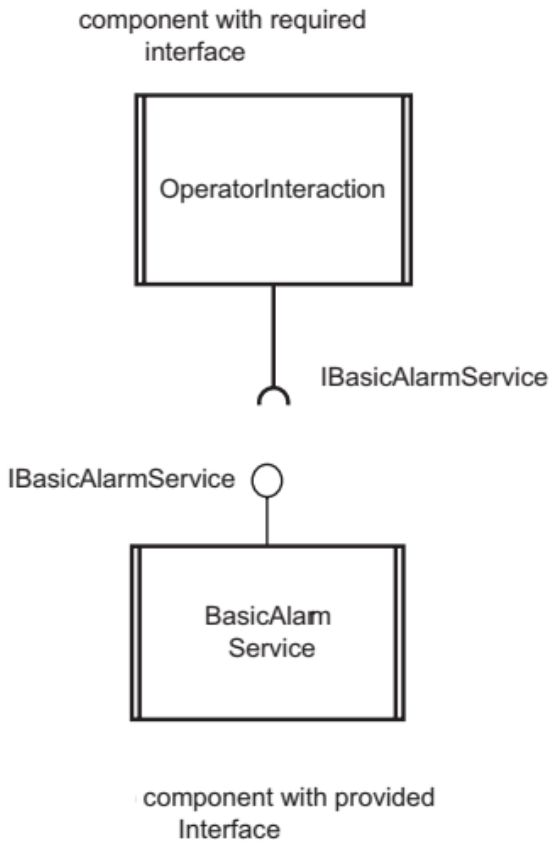
Example of class with public interface and private attributes

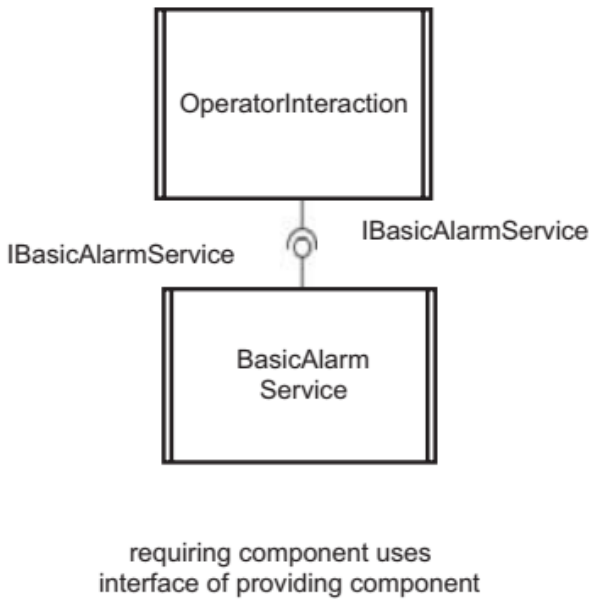
Interface design

- it is useful to depict the design of the interface separately from the component that realizes (i.e., implements) the interface.
- Furthermore, interfaces can be realized in wider contexts than classes.
  - Thus, interfaces for subsystems, distributed components, and passive classes can all be depicted using the same interface notation.
- An interface can be depicted with a different name from the class or component that realizes the interface.
  - By convention, the name starts with the letter “I”
- In UML,
  - an interface can be modeled separately from a component that realizes the interface.
  - An interface can be depicted in two ways: simple and expanded.

Interface design

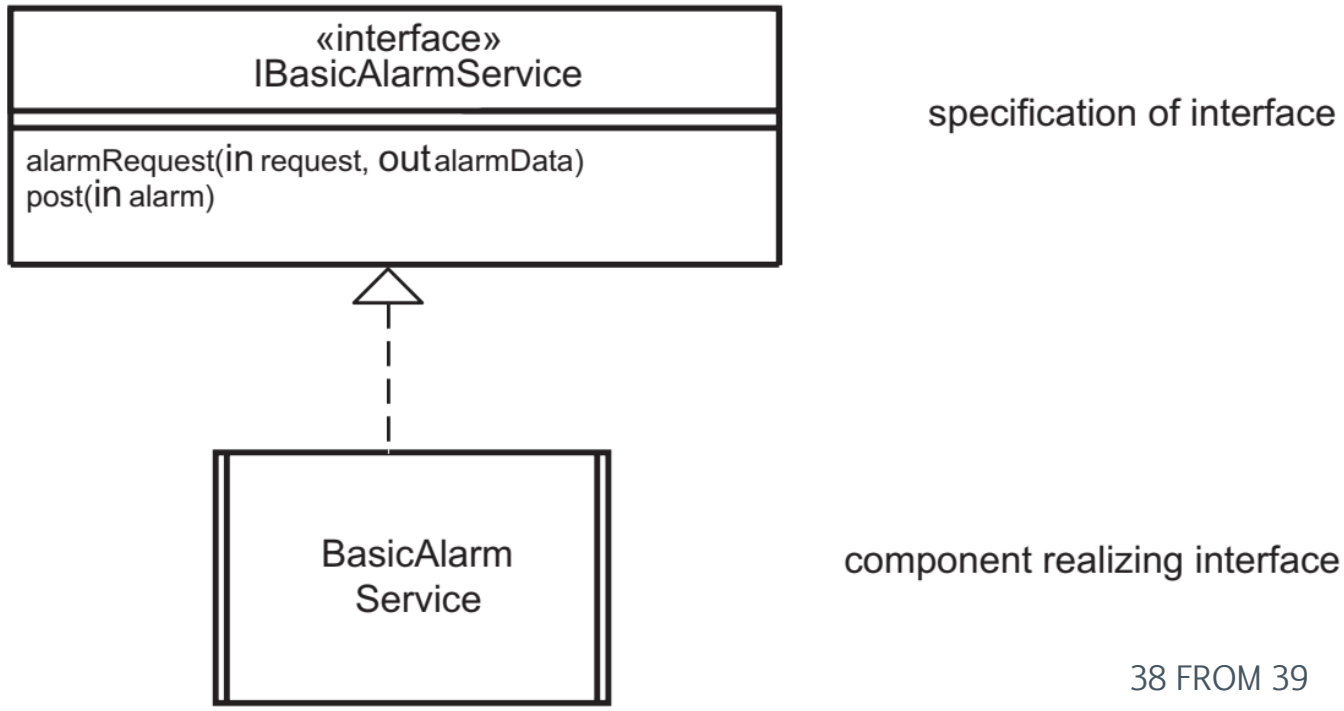
1. In the simple case
   - the interface is depicted as a little circle with the interface name next to it.
   - The class or component that provides the interface is connected to the small circle,





Interface design

- In the expanded case
  - the interface is depicted in a rectangular box with the static modeling notation, with the stereotype «interface» and the interface name in the first compartment
  - The operations of the interface are depicted in the third compartment. The second compartment is left blank



Question?

Bioinformation.ir

info@Bioinformation.ir