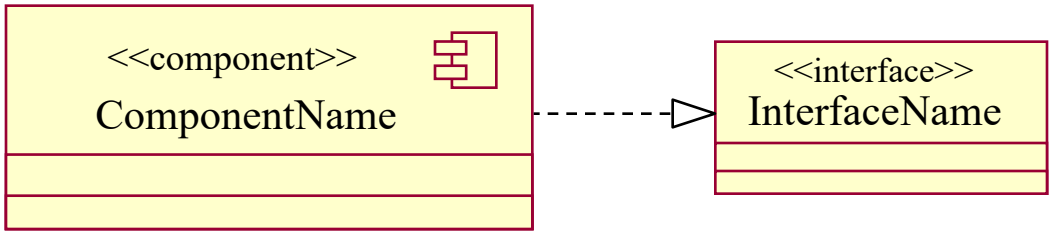
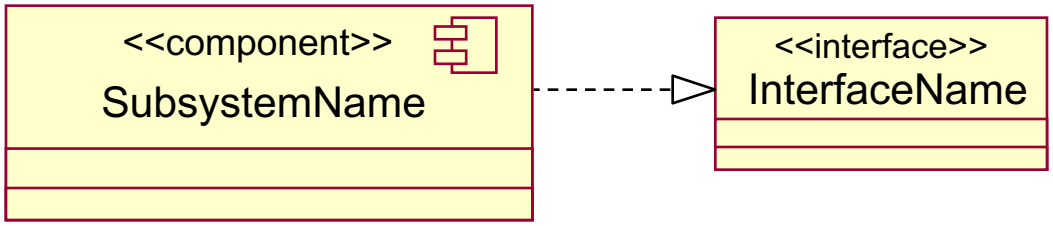
UML Components

- A modular part of a system that hides its contents and whose appearance is replaceable within its environment
  - Defines its behavior in terms of provided and required interfaces that can be wired together
  - Can be replaced at design time or run-time by a component that offers equivalent functionality based on compatibility of its interfaces.



What is a Subsystem?

- A part of a system that encapsulates behavior, exposes a set of interfaces, and contains other model elements.
  - Modeled as a component



OO Principles: Encapsulation and Modularity

Service Oriented Architecture (SOA)

- A SOA is a distributed software architecture that consists of multiple autonomous *services*
- The services are distributed such that they can execute on different nodes with different service providers
- With a SOA, the goal is to develop software *applications that are composed of distributed services*, such that individual services can execute on different platforms and be implemented in different languages

SOA protocols

- Standard *Internet-based protocols* are provided to allow services to communicate with each other and to exchange information
- Each service has a *service description*, which allows applications to discover and communicate with the service
- The service description defines the name of the service, the location of the service, and its data exchange requirements

Service providers and consumers

- A service *provider* supports services used by multiple clients
- Unlike client/server architectures, SOAs build on the concept of *loosely coupled services* that can be discovered and linked to by *clients* (also referred to as service *consumers* or service *requesters*) with the assistance of service *brokers*

SOA design concepts

- An important goal of SOA is to design services as *autonomous reusable components*
- Services are intended to be self-contained and loosely coupled, meaning that dependencies between services are kept to a minimum
- Instead of one service depending on another, *coordination services* are provided in situations in which multiple services need to be accessed and access to them needs to be sequenced
- Several software architectural patterns are described for service-oriented applications:
  - *Broker* patterns, including Service Registration, Service Brokering, and Service Discovery
  - *Transaction* patterns, including Two-Phase Commit, Compound, and Long-Living Transaction patterns
  - and *Negotiation* patterns

Services Design Principles

- Loose coupling
- Service contract
- Autonomy
- Abstraction
- Reusability
- Composability
- Statelessness
- Discoverability

Software Architectural Broker Patterns

- In a SOA, *object brokers* act as intermediaries between clients and services
- In the **Broker** pattern (which is also known as the *Object Broker* or *Object Request Broker* pattern), the **broker** acts as an intermediary between the clients and services
- Services register with the broker
- Clients locate services through the broker
- After the broker has brokered the connection between client and service, communication between client and service can be direct or via the broker

Transparency

- The broker provides both location transparency and platform transparency
- **Location transparency** means that if the service is moved to a different location, clients are unaware of the move and only the broker needs to be notified
- **Platform transparency** means that each service can execute on a different hardware/software platform and does not need to maintain information about the platforms that other services execute on

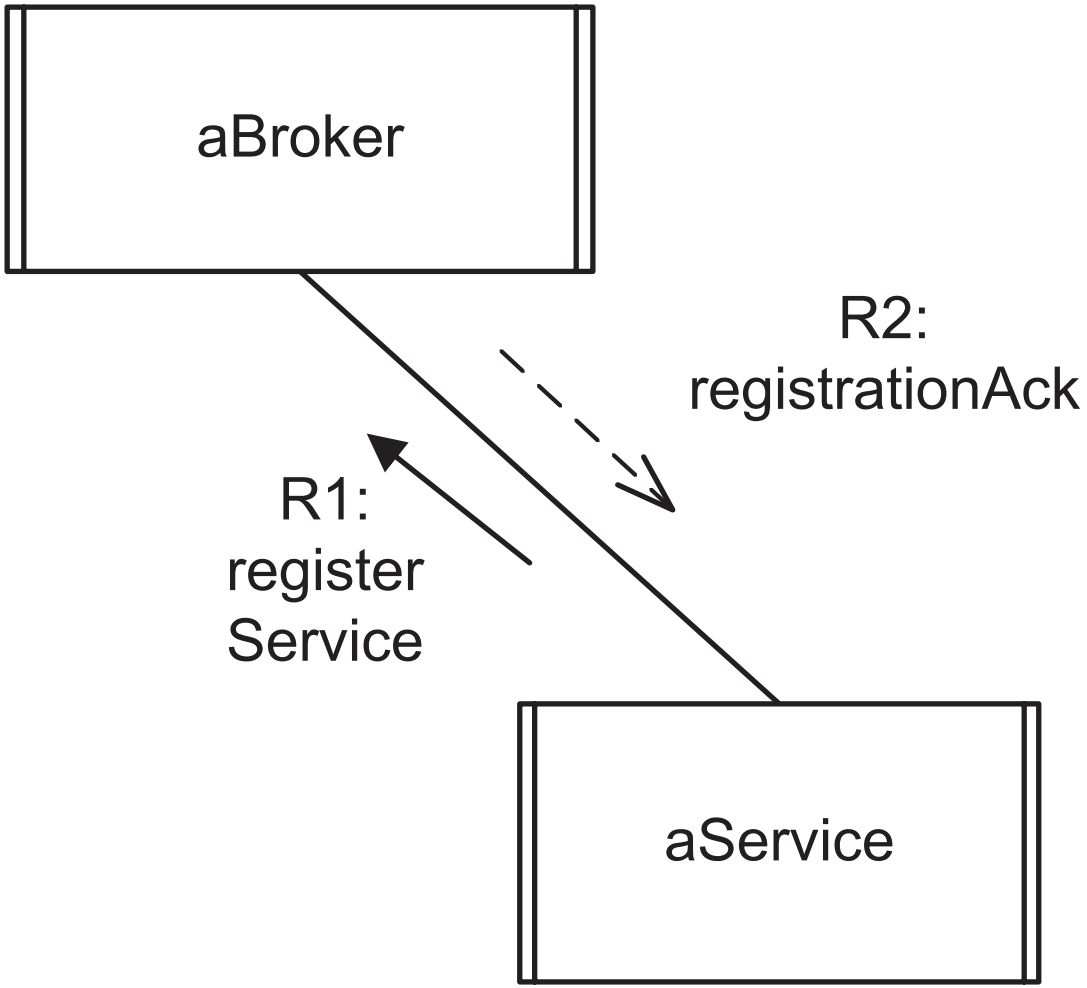
Brokered communication

- With brokered communication, instead of a client having to know the location of a given service, the client queries the broker for services provided
- First, the service must register with a broker as described by the *Service Registration pattern*

Service Registration Pattern

- The service needs to register service information with the broker, including the service name, a description of the service, and the location at which the service is provided
- Message sequence:
  1. the service sends a register service request to the broker
  2. the broker registers the service in the service registry and sends a registration acknowledgment to the service.

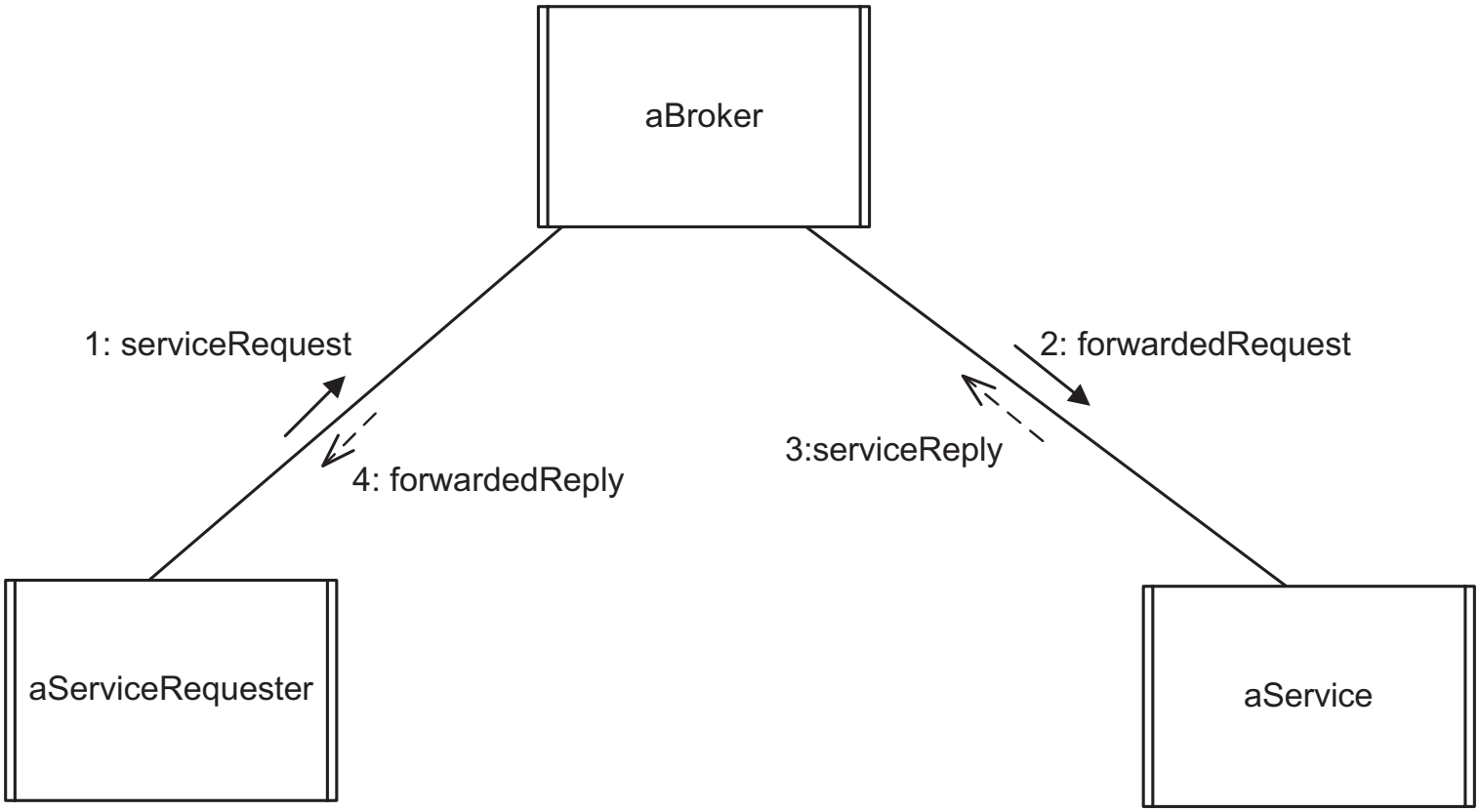
Service Registration Pattern



Broker Fowarding Pattern (white pages)

1. A client sends a message identifying the service required – for example, to withdraw cash from a given bank
2. The broker receives the client request, determines the location of the service (the ID of the node the service resides on), and forwards the message to the service at the specific location
3. The message arrives at the service, and the requested service is invoked
4. The broker receives the service response and forwards it back to the client

Broker Fowarding Pattern (white pages)



Broker Handle Pattern (white pages)

- The **Broker Handle** pattern keeps the benefit of location transparency while adding the advantage of reducing message traffic
- Instead of forwarding each client message to the service, the broker returns a service handle to the client, which is then used for direct communication between client and service
- This pattern is particularly useful when the client and service are likely to have a dialog and exchange several messages between them.

Broker Handle Pattern (white pages)