CSCC01 – Introduction to Software Engineering

Software Architecture

Winter 2024

What is Software Architecture?

“The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.”

Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Architectural Structures

- An architectural structure is a set of elements held together by a relation.
  - It supports reasoning about an attribute of the system that is important to some stakeholder.
- Three categories of structures
  - ***Module*** structures: represent a partitioning of the system into implementation units or modules (e.g. database module, user interface module, etc)
  - ***Component-and-connector*** structures: represent runtime components and communication vehicles among them
  - ***Allocation*** structures: represent the relationship between the system and the non-software structures in its environment (e.g. hardware, development teams, etc)

Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Architectural Patterns

- An architectural pattern establishes a relationship between a ***context***, a ***problem***, and a ***solution***
- It is often based on quality attributes that must be met. Examples include:
  - Modifiability
  - Availability
  - Interoperability
  - Scalability
  - Reusability
  - Performance

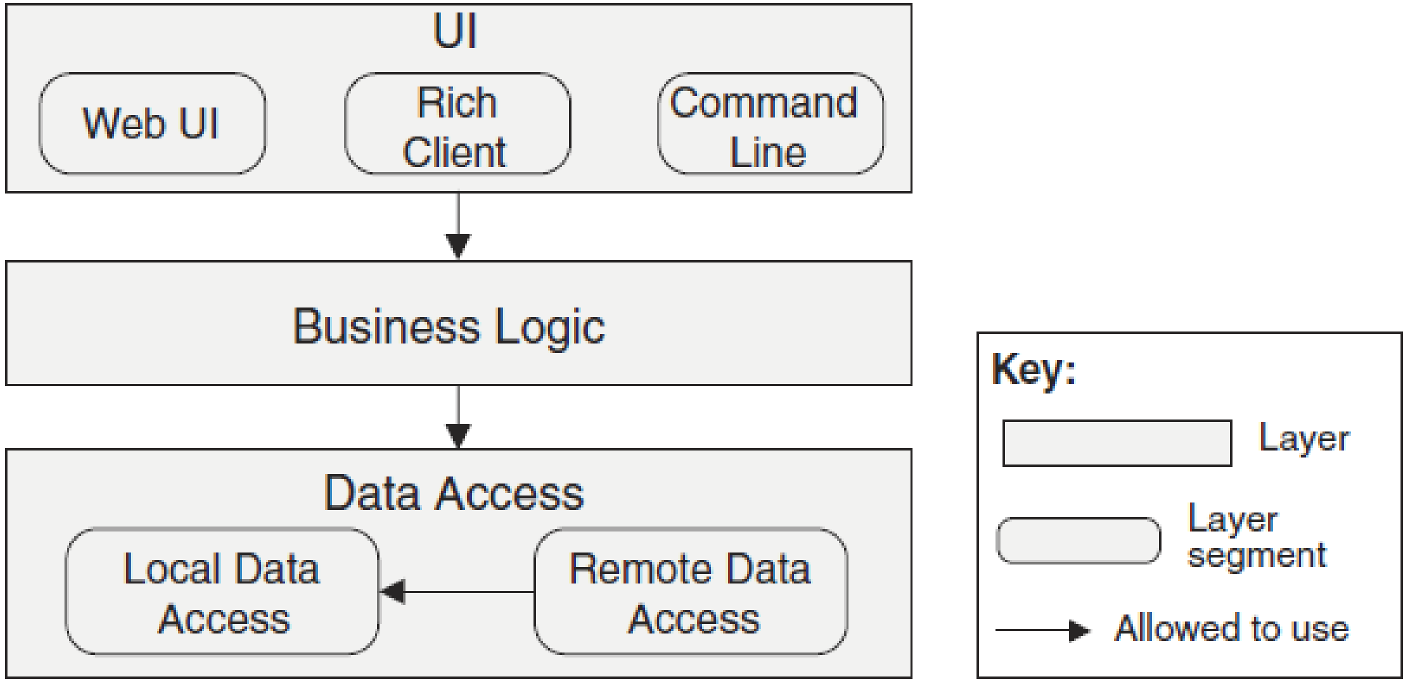
Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Layered Pattern

- Software is divided into layers
  - Each layer is a grouping of modules that offers a cohesive set of services.
  - Every piece of software is allocated to exactly one layer
  - Closed architecture: only next-lower-layer uses are allowed
  - Open architecture: a layer can use services from any lower layer
- Promotes modifiability and reusability
- Challenges
  - Up-front cost and complexity
  - Layers contribute a performance penalty

Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Layered Pattern (Example)



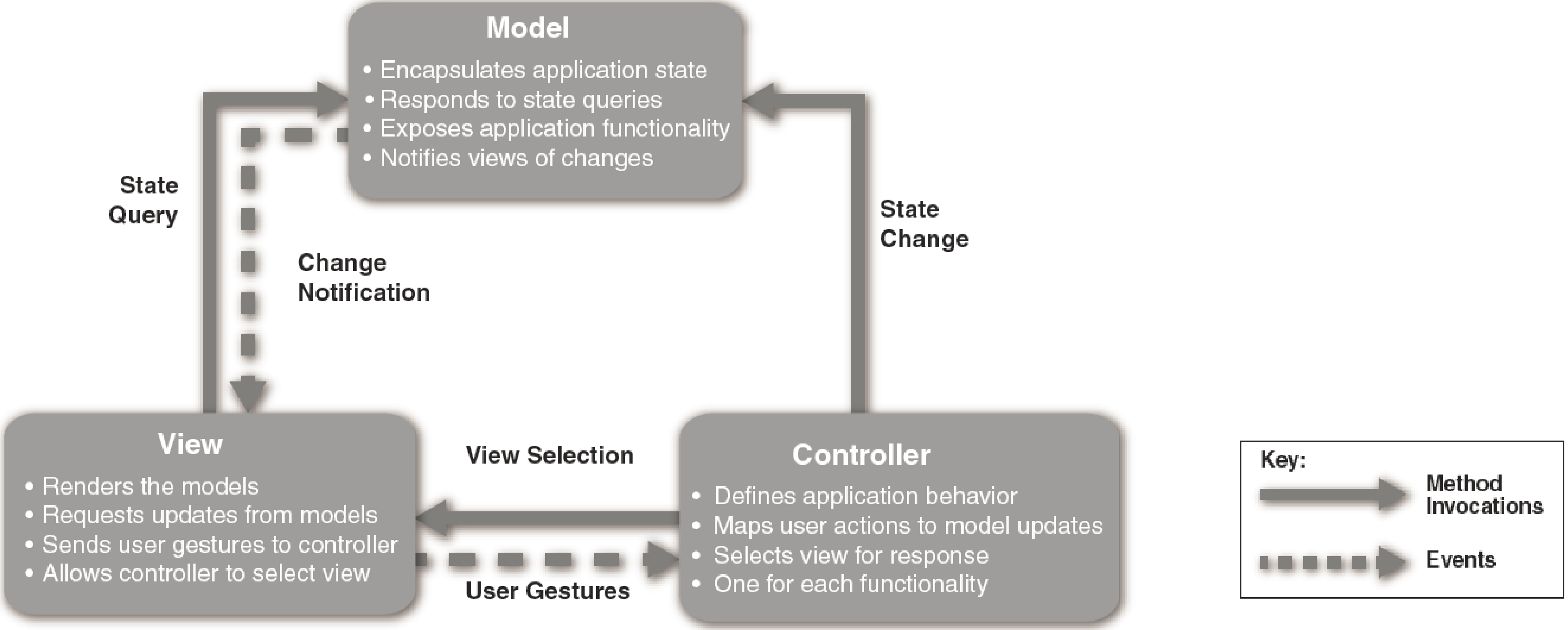
Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Model-View-Controller (MVC) Pattern

- User interface (UI) software is typically the most frequently modified portion of an interactive application.
  - How can UI functionality be kept separate from application functionality and yet still be responsive to user input, or to changes in the underlying application's data?
- In MVC, functionality is separated into three components
  - **Model**: manages the application data
  - **View**: produces a representation of the model
  - **Controller**: translates user actions into changes to the model or changes to the view
- There are several variants of MVC (e.g. MVP)
- Challenges
  - Increased codebase complexity
  - Not suitable for all UI applications

Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

MVC Pattern



Model

- Encapsulates application state
- Responds to state queries
- Exposes application functionality
- Notifies views of changes

View

- Renders the models
- Requests updates from models
- Sends user gestures to controller
- Allows controller to select view

Controller

- Defines application behavior
- Maps user actions to model updates
- Selects view for response
- One for each functionality

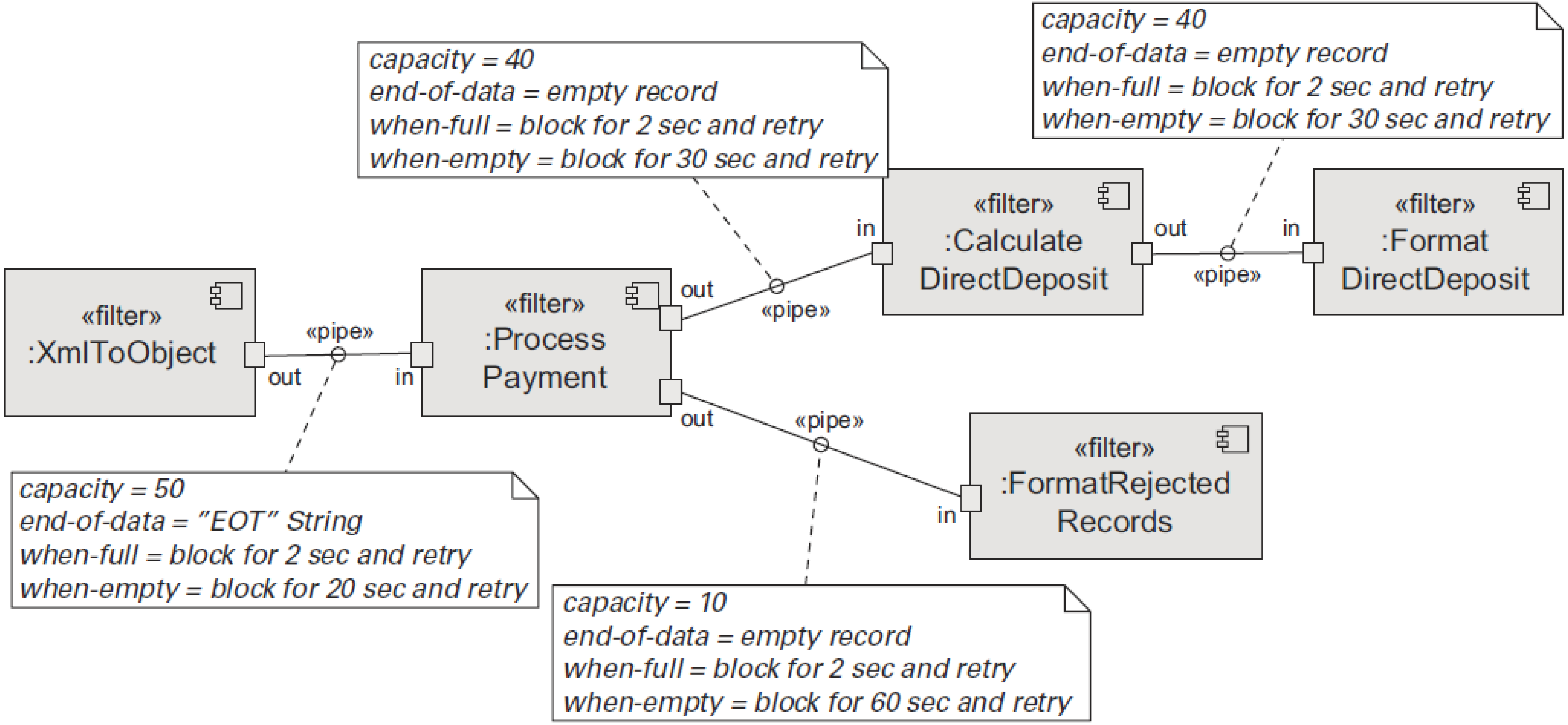
Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Pipe-and-Filter Pattern

- ❑ Many systems are required to transform streams of data from input to output.
- ❑ Such systems need to be divided into reusable, loosely coupled components with simple, generic interaction mechanisms.
- ❑ In a *Pipe-and-Filter* architecture, data undergoes a series of transformations performed by a set of filters connected by pipes
  - Filters typically do not know the identity of their upstream or downstream filters
- ❑ Challenges
  - Not suitable for interactive systems
  - Computational overhead

Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Pipe-and-Filter Pattern (Example)



capacity = 40
end-of-data = empty record
when-full = block for 2 sec and retry
when-empty = block for 30 sec and retry

capacity = 40
end-of-data = empty record
when-full = block for 2 sec and retry
when-empty = block for 30 sec and retry

capacity = 50
end-of-data = "EOT" String
when-full = block for 2 sec and retry
when-empty = block for 20 sec and retry

capacity = 10
end-of-data = empty record
when-full = block for 2 sec and retry
when-empty = block for 60 sec and retry

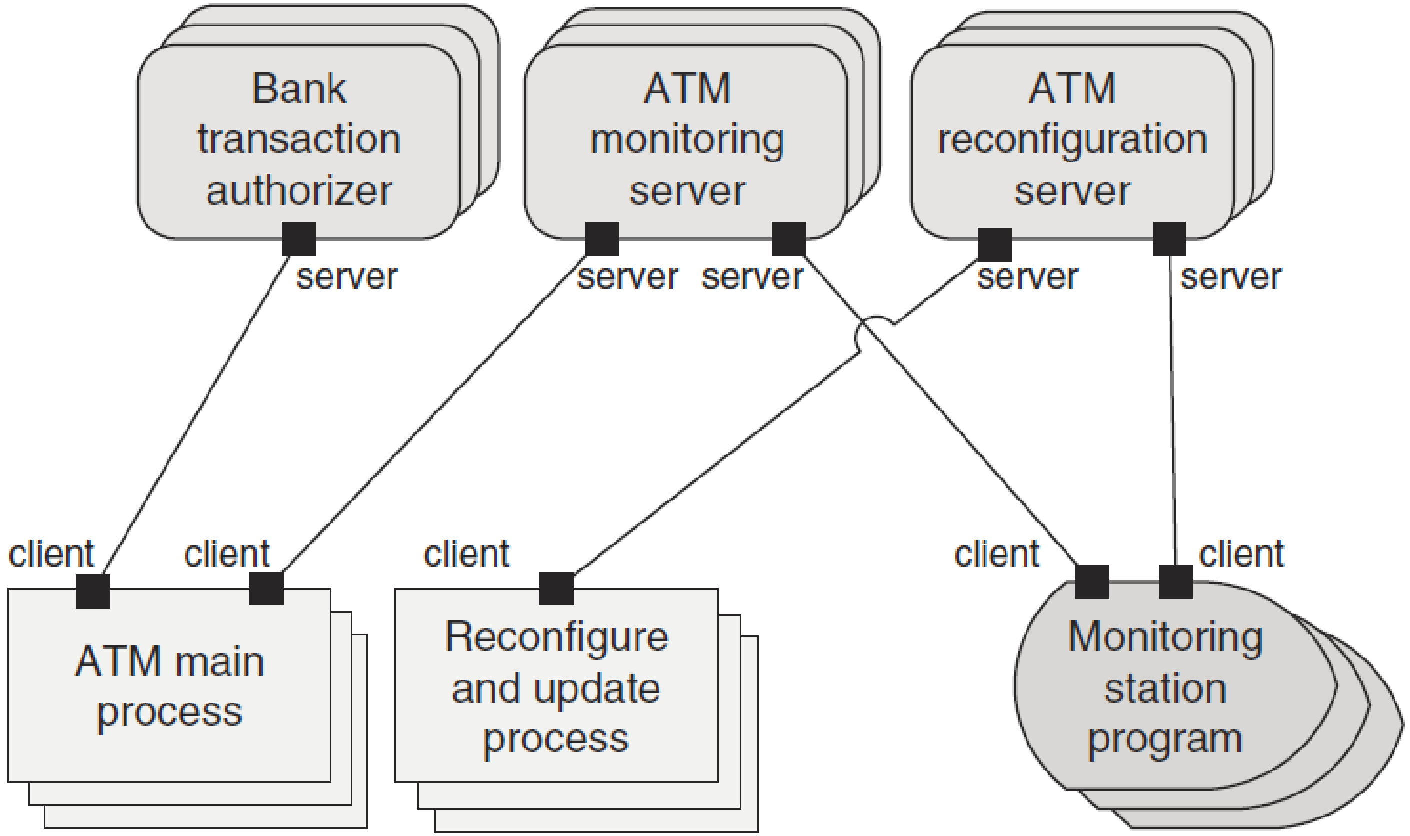
Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Client-Server Pattern

- ❑ In many cases, there are shared resources and services that large numbers of distributed clients wish to access.
  - How to manage the shared resources and services while promoting modifiability and reuse?
- ❑ In a *Client-Server* architecture, clients interact by requesting services of servers, which provide a set of services.
  - Some components may act as both clients and servers.
  - There may be one central server or multiple distributed ones.
- ❑ Challenges
  - The server can be a performance bottleneck
  - Single point of failure

Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Client-Server Pattern (Example)


Bank transaction authorizer
ATM monitoring server
ATM reconfiguration server
server
server
server
server
server
client
client
client
client
client
ATM main process
Reconfigure and update process
Monitoring station program

Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

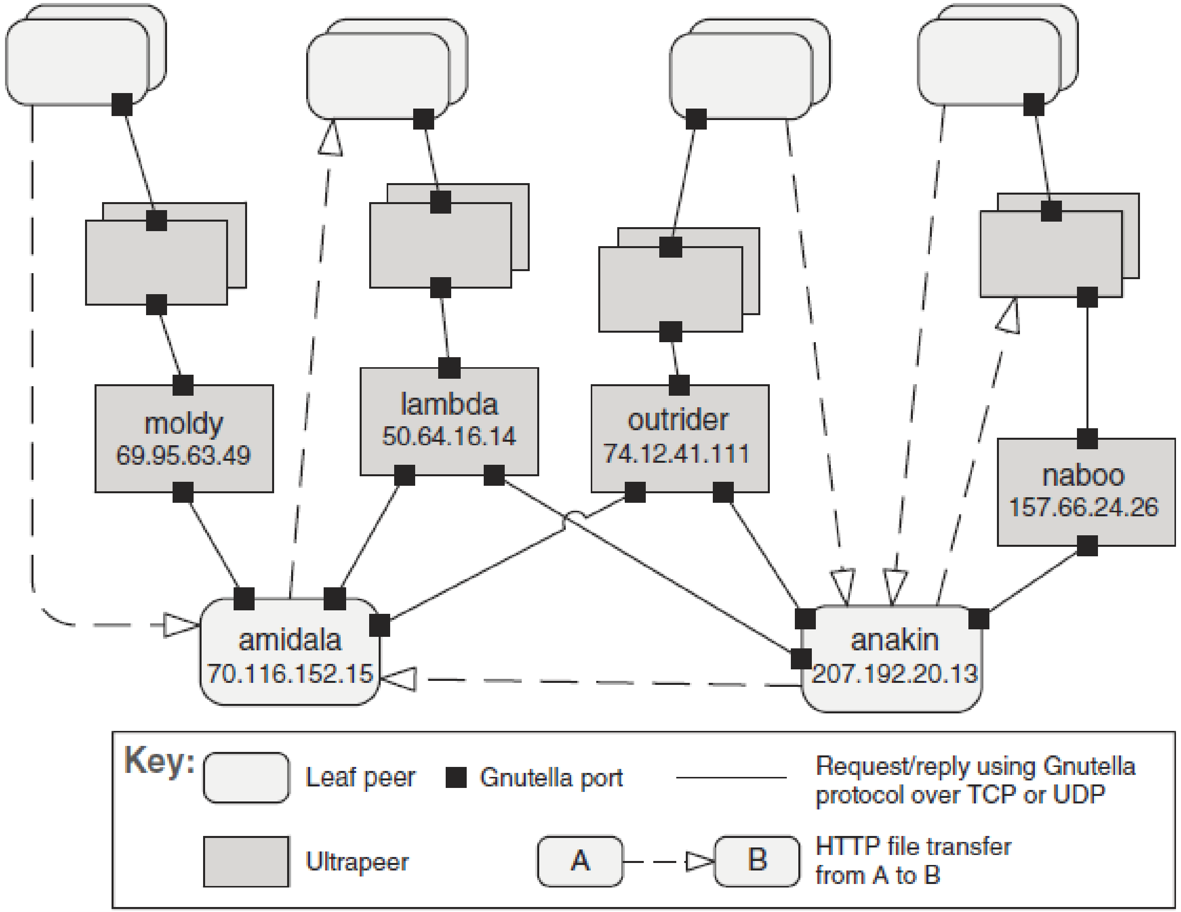
Peer-to-Peer Pattern

- Peer-to-peer communication is typically a request/reply interaction without the asymmetry found in the client-server pattern.
  - Any component can, in principle, interact with any other component by requesting its services.
  - A peer-to-peer architecture may have specialized peer nodes (called supernodes) that have indexing or routing capabilities.

- Challenges
  - Managing issues such as security and data consistency
  - No guarantees in terms of quality goals

Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Peer-to-Peer Pattern (Example)

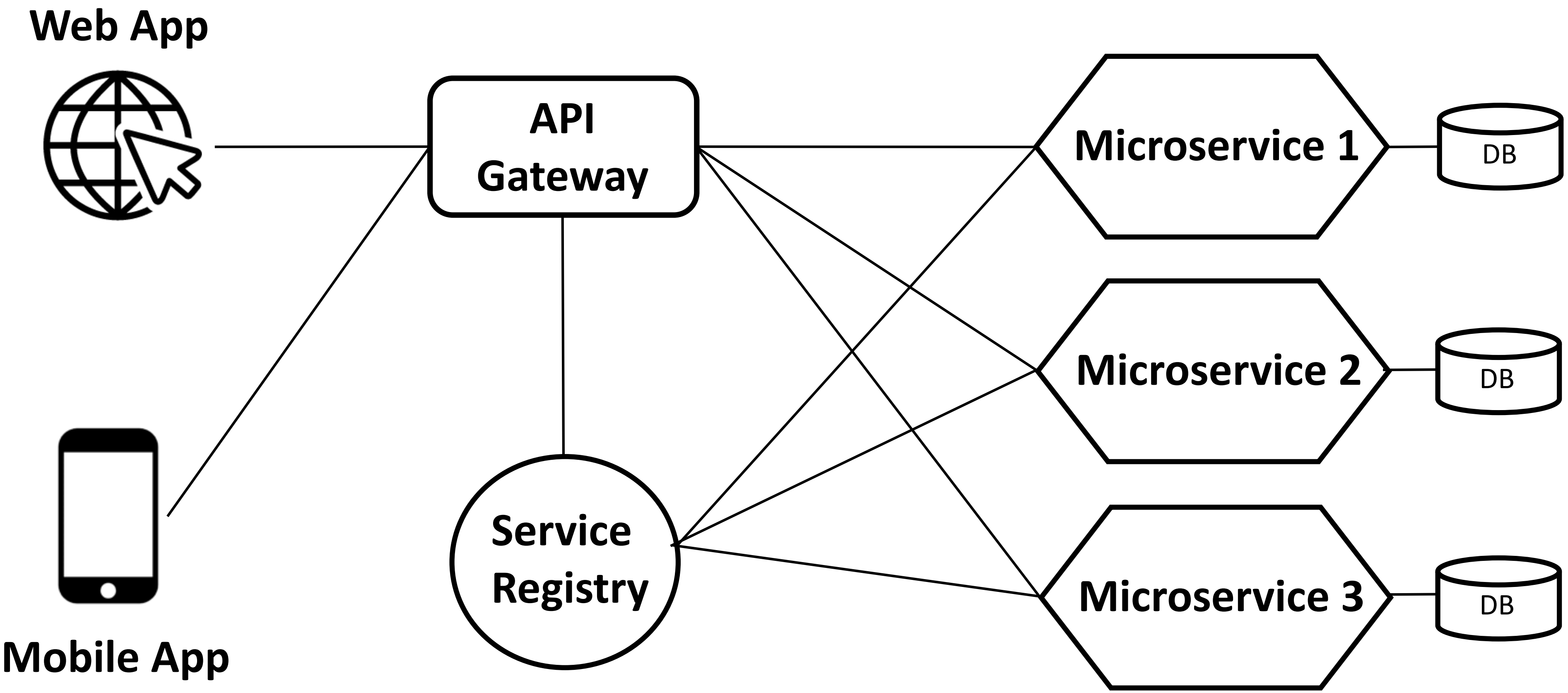


Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd. ed.). Addison-Wesley Professional.

Microservices

- In a *Microservices* architecture, an application is decomposed into small, modular services that are developed and deployed independently
  - This promotes modifiability, reusability, interoperability, and scalability
- These services typically communicate through RESTful APIs with HTTP as the underlying communication protocol
- A gateway can be used to facilitate communication between external clients and the distributed microservices
- Microservices usually register themselves with a service registry so they can be discovered
- Challenges
  - Design and implementation
  - Communication
  - Testing the application as a whole

Microservices (Example)



REST

- REST stands for Representational State Transfer, an architectural style for designing networked applications
- RESTful communication is stateless, i.e. each request from a client to a server contains all the information needed for processing
- Resources can be data or services
  - Identified by unique Uniform Resource Identifiers (URIs)
  - Represented in a format such as JSON or XML

HTTP (Hypertext Transfer Protocol)

- In HTTP communication, the client sends a request and the server responds with the requested data.
- HTTP is stateless: each request from a client to a server is independent and does not rely on previous requests.
- HTTP defines standard methods for interacting with resources
  - **GET**: Retrieve data from the server
  - **POST**: Send data to the server to create a new resource
  - **PUT**: Update an existing resource on the server
  - **DELETE**: Request the removal of a resource on the server
- HTTP status codes indicate the outcome of a request. Examples include:
  - **200 OK**: Successful request
  - **404 Not Found**: Requested resource not found