Software Architecture Theory

# P09. *Architecture Patterns*

2014

Sungwon Kang

# 9. 아키텍처 패턴

9.1 정의 및 기술방법

9.2 아키텍처 패턴의 종류

9.3 아키텍처 패턴의 적용

9.4 아키텍처 스타일과 아키텍처 패턴의 비교

9.5 아키텍처 패턴과 디자인 패턴의 비교

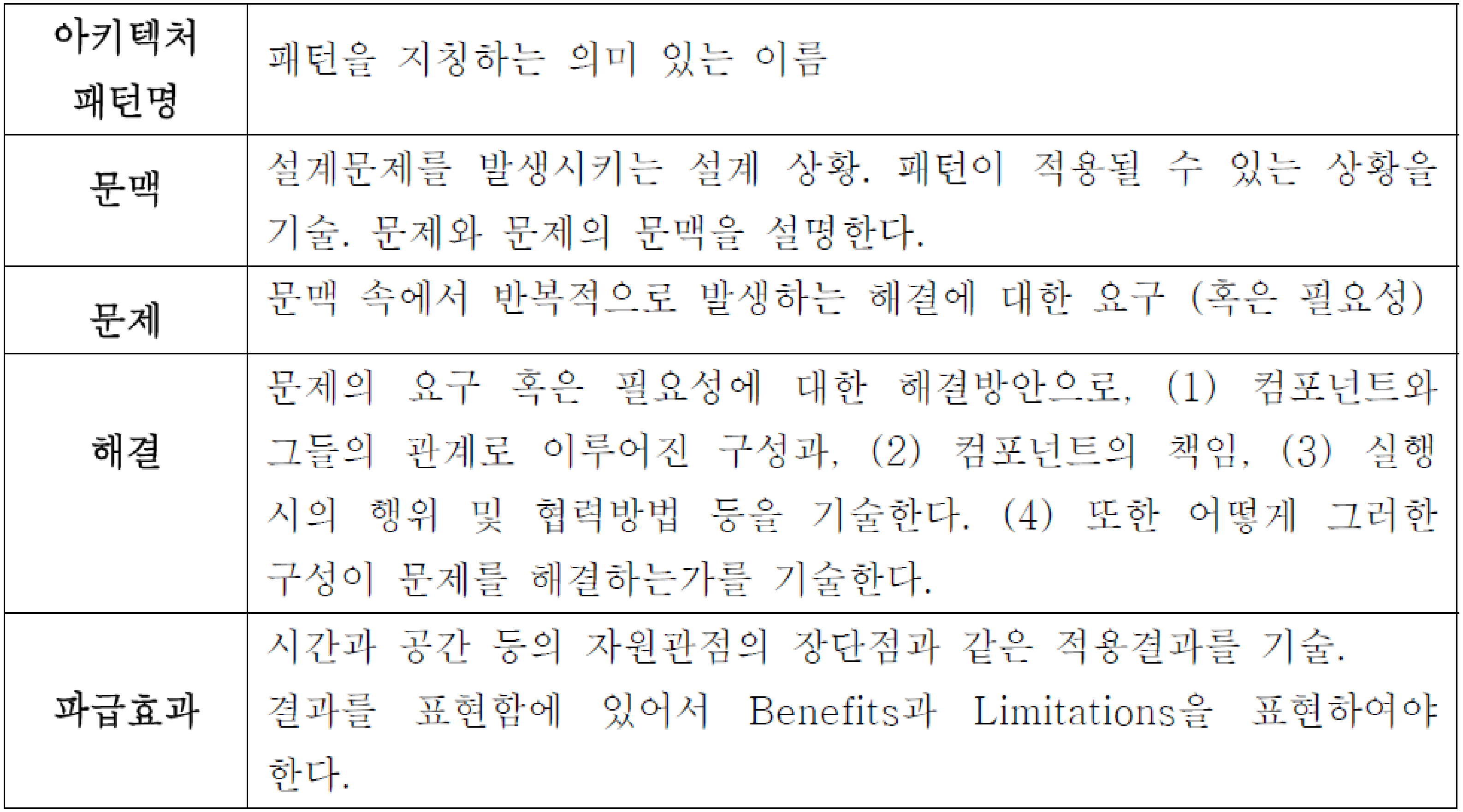
# 9.1 정의 및 기술방법

## Definition of Architectural Pattern [Taylor 09]

**Definition :** An **architectural pattern** is a named collection of architectural design decisions that are applicable to a recurring design problem parameterized to account for different software development contexts in which that problem appears.

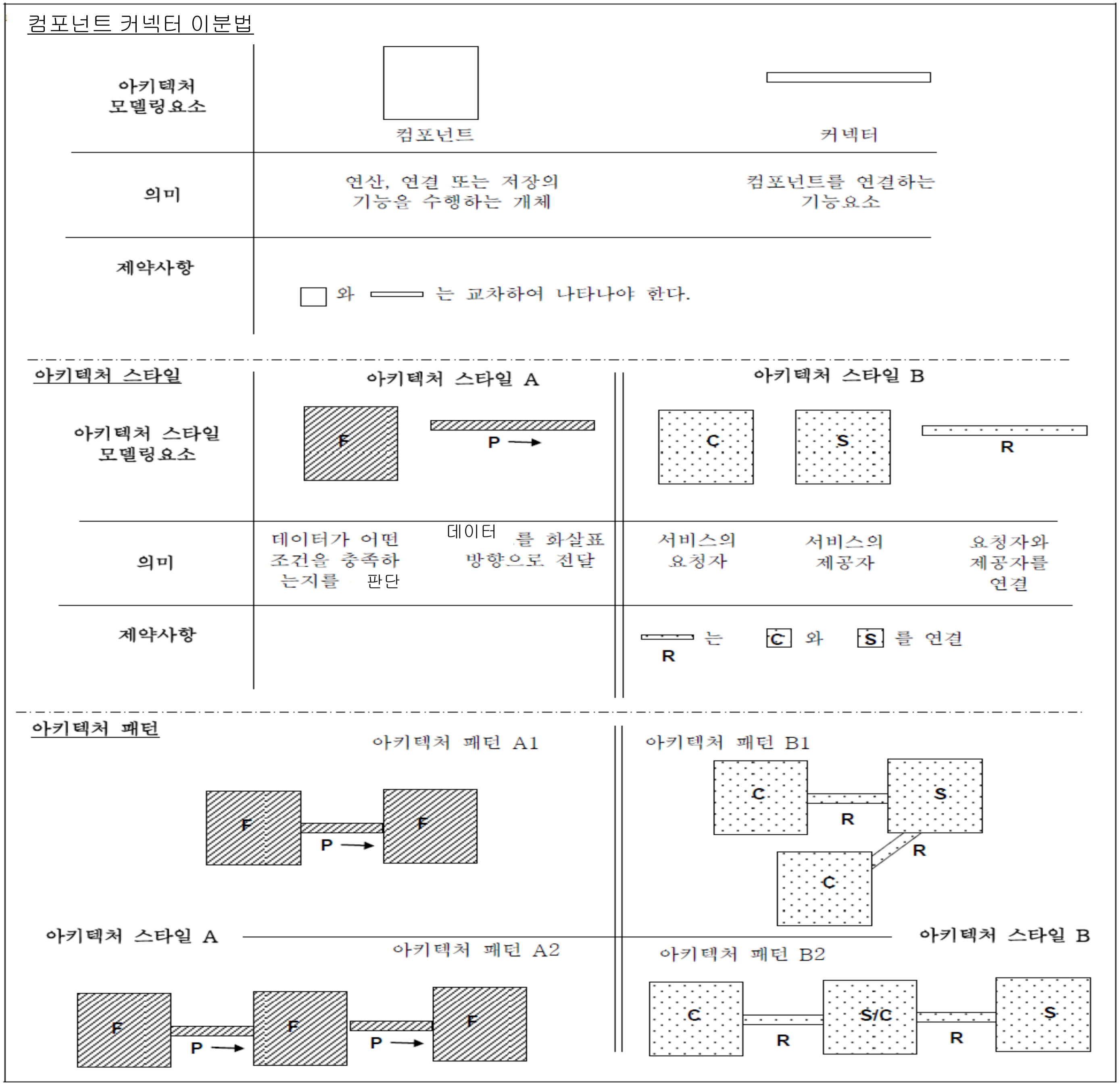
- **Provides a set of specific design decisions that have been identified as effective for organizing certain classes of software systems, or more typically, specific subsystems.**

- **Can be configurable in that they need to be instantiated with the components and connectors particular to an application.**



| | |
|---|---|
| 아키텍처 패턴명 | 패턴을 지칭하는 의미 있는 이름 |
| 문맥 | 설계문제를 발생시키는 설계 상황. 패턴이 적용될 수 있는 상황을 기술. 문제와 문제의 문맥을 설명한다. |
| 문제 | 문맥 속에서 반복적으로 발생하는 해결에 대한 요구 (혹은 필요성) |
| 해결 | 문제의 요구 혹은 필요성에 대한 해결방안으로, (1) 컴포넌트와 그들의 관계로 이루어진 구성과, (2) 컴포넌트의 책임, (3) 실행 시의 행위 및 협력방법 등을 기술한다. (4) 또한 어떻게 그러한 구성이 문제를 해결하는가를 기술한다. |
| 파급효과 | 시간과 공간 등의 자원관점의 장단점과 같은 적용결과를 기술. 결과를 표현함에 있어서 Benefits과 Limitations을 표현하여야 한다. |

그림 9-1. 아키텍처패턴의 표현



컴포넌트 커넥터 이분법

아키텍처 모델링요소: 컴포넌트 / 커넥터

의미: 연산, 연결 또는 저장의 기능을 수행하는 개체 / 컴포넌트를 연결하는 기능요소

제약사항: □ 와 ▭ 는 교차하여 나타나야 한다.

아키텍처 스타일

아키텍처 스타일 A / 아키텍처 스타일 B

아키텍처 스타일 모델링요소

의미: 데이터가 어떤 조건을 충족하는지를 판단 / 데이터를 화살표 방향으로 전달 / 서비스의 요청자 / 서비스의 제공자 / 요청자와 제공자를 연결

제약사항: ▭ R 는 C 와 S 를 연결

아키텍처 패턴

아키텍처 패턴 A1

아키텍처 패턴 B1

아키텍처 스타일 A

아키텍처 패턴 A2

아키텍처 스타일 B

아키텍처 패턴 B2

그림 9-2. 아키텍처패턴의 표현

# 9.2 아키텍처 패턴의 종류

Broker
MVC
PAC
Microkernel
Reflection
Client-Dispatcher-Server
Layers
Decomposition
Pipe and Filter
Publish-Subscribe
Client-Server
Blackboard

[Buschmann 96]

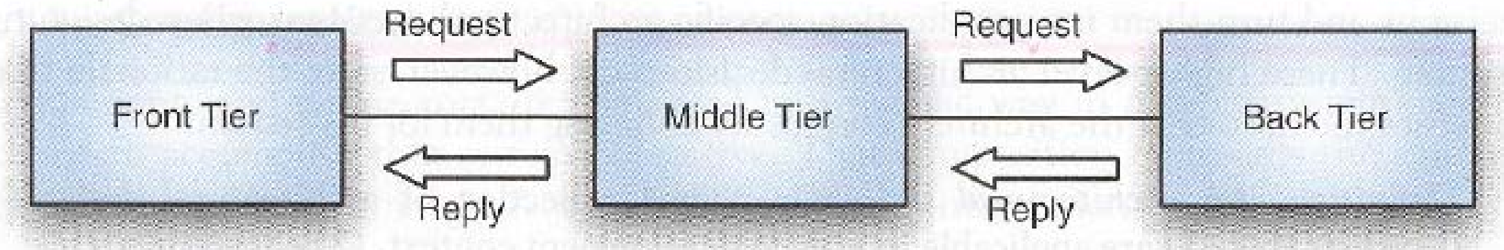
# 표 9-1. 분산컴퓨팅을 위한 아키텍처 패턴의 분류 (출처: [Buschmann 07a])

| 패턴의 종류 | 패턴 |
|---|---|
| 진흙에서 구조로 (From Mud to Structure) | Domain Model, Layers, Model-View-Controller, Presentation-Abstraction-Control, Microkernel, Reflection, Pipes and Filters, Shared Repository, Blackboard, Domain Object, |
| 분산 인프라 (Distribution Infrastructure) | Messaging, Message Channel, Message Endpoint, Message Translator, Message Router, Publisher-Subscriber, Broker, Client Proxy, Requestor, Invoker, Client Request Handler, Server Request Handler |
| 이벤트 집결 및 발송(Event Demultiplexing and Dispatching) | Reactor, Proactor, Acceptor-Connector, Asynchronous Completion Token |
| 인터페이스 분리(Interface Partitioning) | Explicit Interface, Extension Interface, Introspective Interface, Dynamic Invocation Interface, Proxy, Business Delegate, Facade, Combined Method, Iterator , Enumeration Method, Batch Method |
| 컴포넌트 분리 (Component Partitioning) | Encapsulated Implementation, Whole-Part, Composite, Master-Slave, Half-Object plus P:rotocol, Replicated Component Group |
| 어플리케이션 제어 (Application Control) | Page Controller, Front Controller, Application controller, command Processor, Template View, Transform View, Firewall Proxy, authorization |

| | |
|---|---|
| 병행성<br>(Concurrency) | Half-Sync/Half-Async, Leader/Followers, active Object, Monitor Object |
| 동기화<br>(Synchronization) | Guarded Suspension, Future, Thread-Safe Interface, Double-Checked Locking, Strategized Locking, Scoped Locking, Thread-Specific Storage |
| 객체상호작용<br>(Object Interaction) | Observer, Double Dispatch, Mediator, Command, Memento, Context Object, Data Transfer Object Message |
| 적응과 확장<br>(Adaption and Extension) | Bridge, Object Adapter, Chain of Responsibility, Interpreter, interceptor, Visitor, Decorator, Execute-Around Object, Template Method, Strategy, Null Object, Wrapper Facade, Declarative Component Configuration |
| 상태적 행위<br>(Modal Behavior) | Objects for States, Methods for States, Collections for States |
| 자원관리<br>(Resource Management) | Container, Component Configurator, Object Manager, Lookup, Virtual Proxy, Lifecycle  Callback, Task Coordinator, Resource Pool, Resource Cache, Lazy Acquisition, Eager Acquisition, Partial Acquisition Activator, Evictor, Leasing, Automated Garbage Collection, Counting Handler, Abstract  Factory, Builder, Factory Method, Disposal Method |
| 데이터베이스 접근<br>(Database Access) | Database Access Layer, Data Mapper, Row Data Gateway, Table Data Gateway, Active Record |

# 9.3 아키텍처 패턴의 적용(Based on [Taylor 09])

**Figure 3-4.** *Graphical view of the three-tier system architectural pattern.*



- **Three tiers**:
  - Font tier (client tier):
  - Middle tier (application tier or business logic tier):
  - Backend (data tier):

- **Interaction among the tiers**
  - Request-reply paradigm
  - Pattern does not prescribe the interactions further but possible interactions are:
    - Synchronous
    - Request-triggered
    - Singe request – single reply
    - Multiple requests – single reply
    - Multiple reply to a single request
    - Periodic updates or
    - Etc.

# Architectural Patterns [Taylor 09]

- To come up with a specific architecture based on the *three tier pattern*, the architect needs to specify:
  1. Which application-specific user interface, processing, and data access and storage facilities are needed and how they should be organized within each tier
  2. Which mechanisms should be used to enable interaction across the tiers

To solve the same problem **using architectural style** requires more attention from the system's architect, and provides less direct support

=> **The three-tier architectural pattern** can be thought of as **two specific architectures** that are designed according to the **client-server style** and overlaid on top of each otter:
**The front tier** is the client to the middle tier while the **middle tier** ties the client to the **back tier**, the middle tier is thus a server in the first client-server architecture and a client in the second.

– System adhering to the client-server style are sometimes referred to as *two tier systems*.

# Architectural Patterns [Taylor 09]

- Three different solutions of the Lunar Lander architecture design.
  1. State-Logic-Display (aka. Three Tier)
  2. Model-View-Controller (for GUI)
  3. Sense-Compute-Control (aka. Sensor-Controller-Actuator)

# (1) State-Logic-Display (Three Tier) Pattern

- Commonly employed in business applications where
  - there is a data store behind a set of business logic rules
  - Business logic is accessed by UI component
- Popular for distributed systems implementation in which communication between the components is by RPC
- Applications:
  - In business applications,
    - S: large database server
    - L: business logic component
    - D: simple component for managing interaction with a user on a desktop PC
  - Multiplayer games: each player has her own display component
  - Many Web-based applications
    - S: database accessed by the Web server on a local host
    - L: Web server plus application specific logic
    - D: user’s Web browser

    ⇒ However, connection between components is by HTTP rather than RPC

Hard to say what specific domains **this pattern** is useful for.

# (1) State-Logic-Display (Three Tier) Pattern

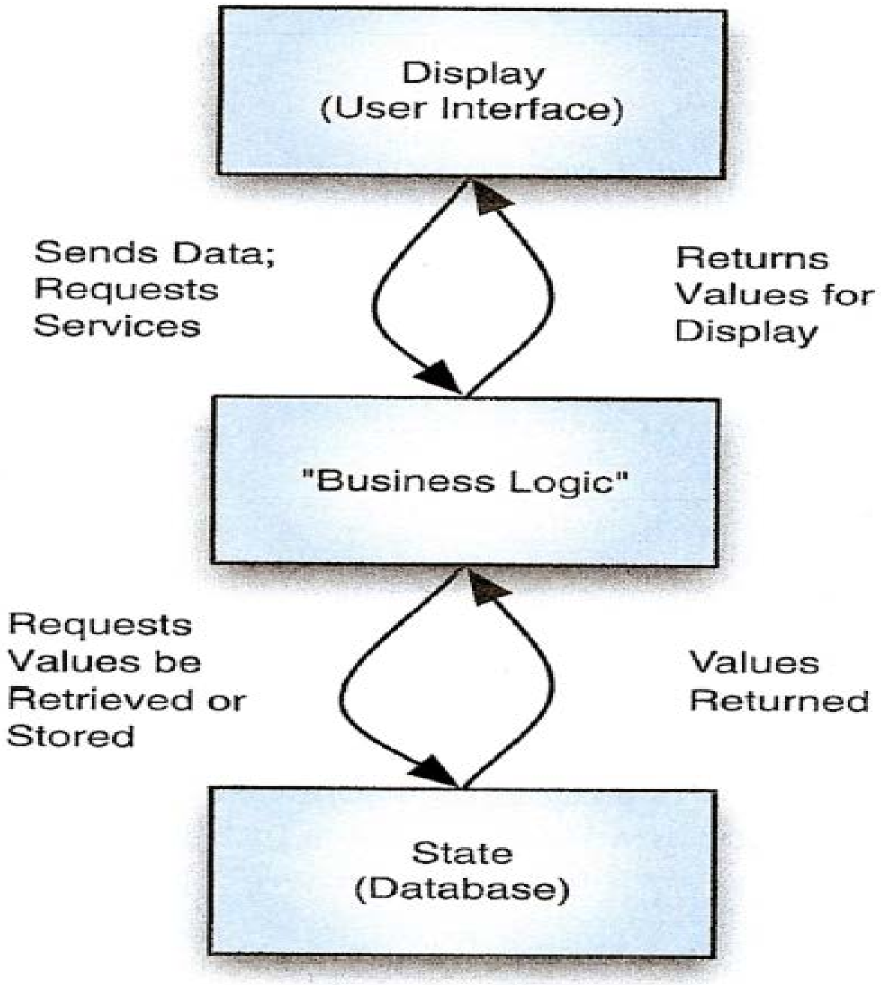

Display
(User Interface)
Sends Data; Requests Services
Returns Values for Display
"Business Logic"
Requests Values be Retrieved or Stored
Values Returned
State
(Database)

**Figure 4-3.** *Canonical form of state-logic-display architectures.*

# (2) Model-View-Controller (MVC) Pattern

- A dominant GUI design pattern since invented in 1980's
- Can be regarded as design pattern ***or*** architectural pattern
- Objectives:
  - promotes separation of information manipulated by a program and depictions of user interactions with that information

    => independent development paths
- **Model component**: encapsulates the information used by the application
- **View component**: encapsulates the information chosen and necessary for graphical depiction of that information
- **Controller component**: encapsulates the logic necessary to maintain consistency between the model and the view, and to handle inputs from the user as they relate to eh depiction

# (2) Model-View-Controller (MVC) Pattern

- Notional **interaction** between these components:

  **(1) When the application changes value in the model object**, notification of that change is sent **to the view** so that any affected parts of the depiction can be updated and redrawn

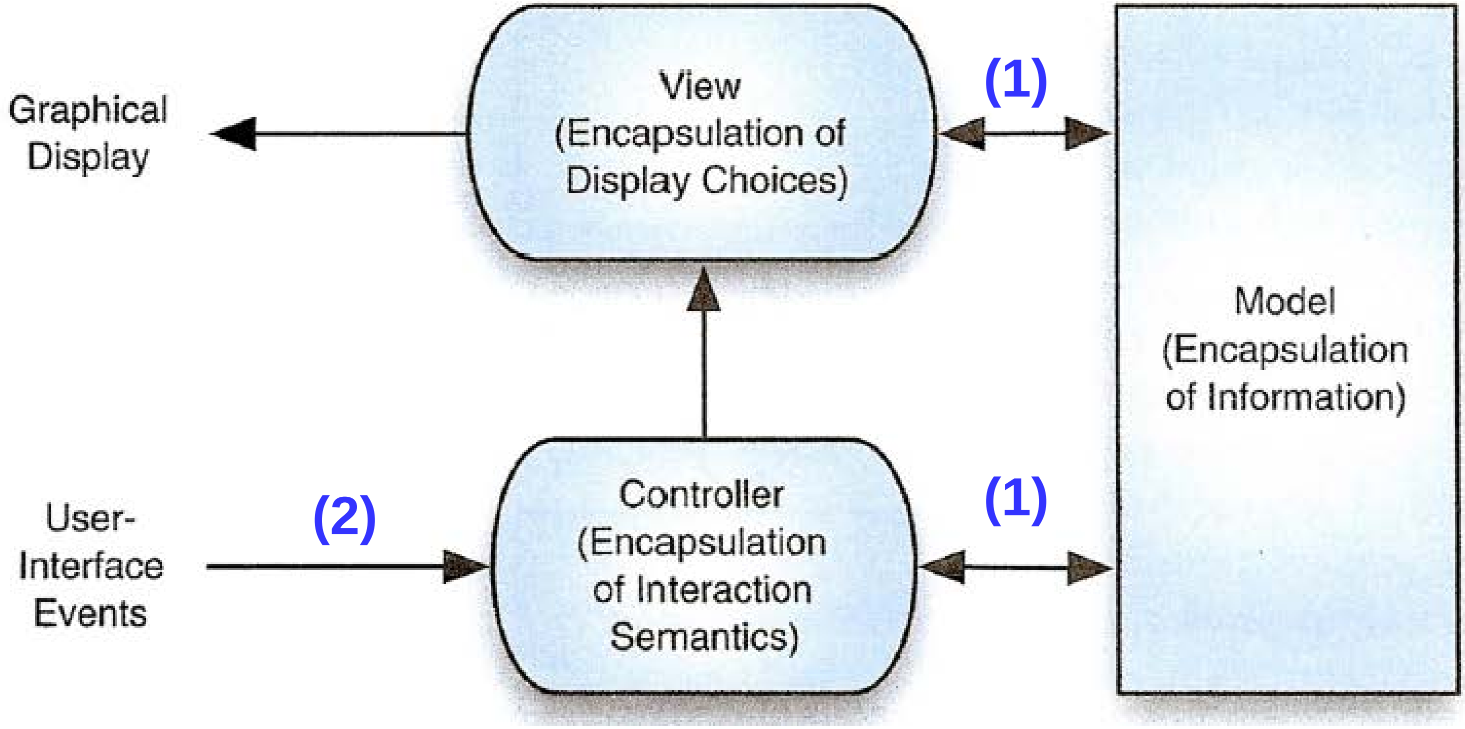
  - Notification also typically goes to the controller as well, so that the controller can modify the view if its logic so requires.

  **(2) When handling input from the user** (such as a mouse click on part of the view), the viewing system sends the user event **to the controller**.

  - The controller then updates the model object in keeping with the desired semantics.

# (2) Model-View-Controller (MVC) Pattern

Figure 4-4.
*Notional model-view-controller pattern.*


Graphical Display
View (Encapsulation of Display Choices)
(1)
Model (Encapsulation of Information)
User-Interface Events
(2)
Controller (Encapsulation of Interaction Semantics)
(1)

# (2) Model-View-Controller (MVC) Pattern

- Can be seen at work in the WWW.
  - Web resources –> model
  - HTML rendering agent within a browser –> view
  - Part of the browser that responds to user input and which causes either interactions with a Web server or modifies the browser's display –> controller

- Inspired the development of PAC (Presentation-Abstraction-Control) pattern

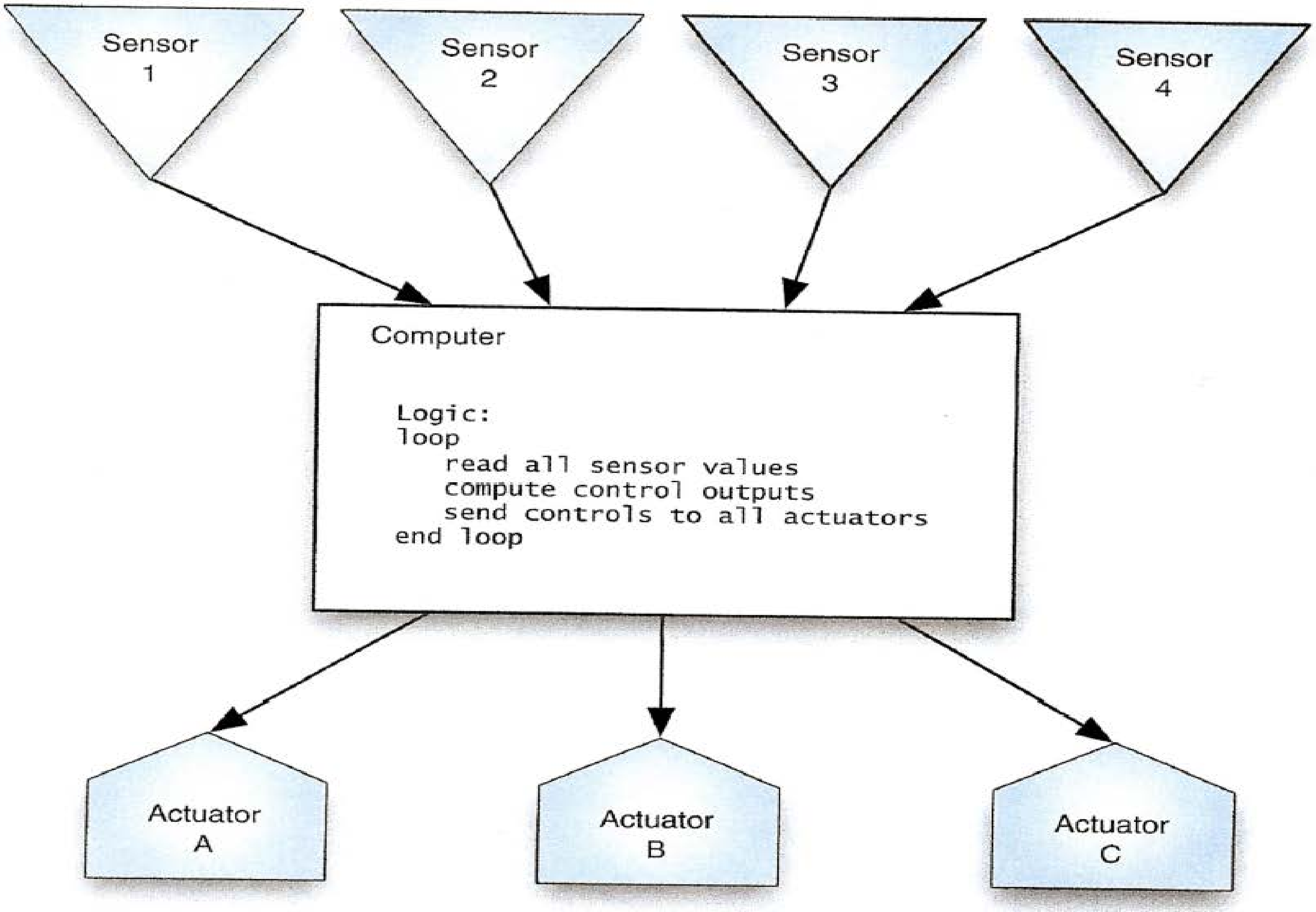
# (3) Sense-Compute-Control Pattern

- Used in structuring embedded control application

  **Example** Kitchen appliance, automotive applications, robotic control

- A **computer** is embedded in some application:
  - **Sensors** from various devices
    - are connected to the computer
    - The computer samples them for values
  - Hardware **actuators**
    - The computer sends a signal to them
- Architectural pattern:
  - Cycling through the steps of
    - reading all the sensor values
    - executing a set of control laws or function and then
    - sending out s to the various actuators
  - The cycle is keyed to a clock
  - There is implicit feedback via external environment

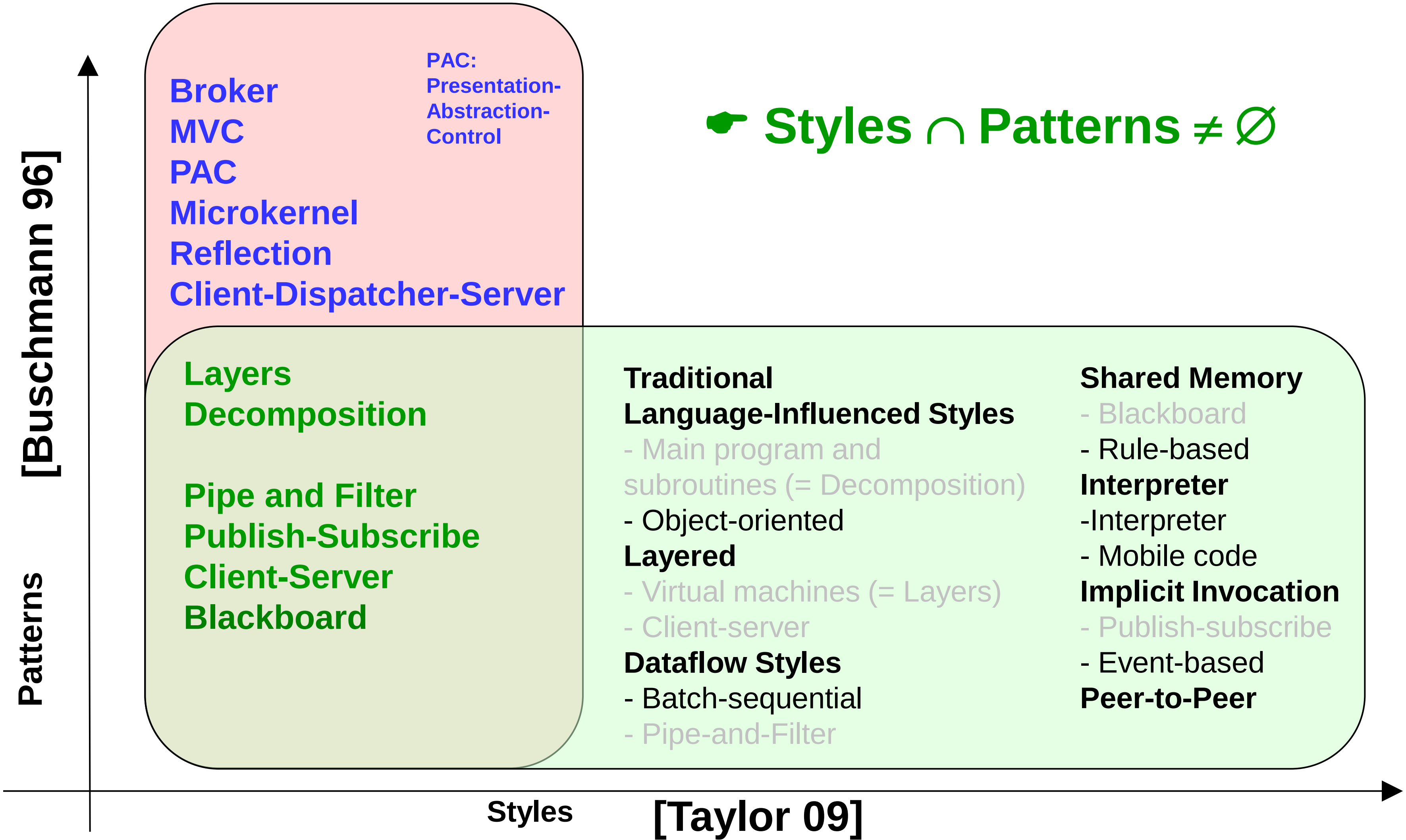
# (3) Sense-Compute-Control Pattern



```
Logic:
loop
   read all sensor values
   compute control outputs
   send controls to all actuators
end loop
```

Figure 4-5. *Sense-compute-control: Different-shaped boxes are used to indicate the different types of devices present in the system.*

# 9.4 아키텍처 스타일과 아키텍처 패턴의 비교



☛ Styles ∩ Patterns ≠ ∅

Broker
MVC
PAC
Microkernel
Reflection
Client-Dispatcher-Server

PAC:
Presentation-Abstraction-Control

Layers
Decomposition

Pipe and Filter
Publish-Subscribe
Client-Server
Blackboard

**Traditional Language-Influenced Styles**
- Main program and subroutines (= Decomposition)
- Object-oriented

**Layered**
- Virtual machines (= Layers)
- Client-server

**Dataflow Styles**
- Batch-sequential
- Pipe-and-Filter

**Shared Memory**
- Blackboard
- Rule-based

**Interpreter**
-Interpreter
- Mobile code

**Implicit Invocation**
- Publish-subscribe
- Event-based

**Peer-to-Peer**

# 9.4 아키텍처 스타일과 아키텍처 패턴의 비교

**Proposed Definitions**

- What influences determination of architecture viewpoint frameworks is architecture style.
- What provides solution within a specific viewpoint is a architecture pattern.

# 9.5 아키텍처 패턴과 디자인 패턴의 비교

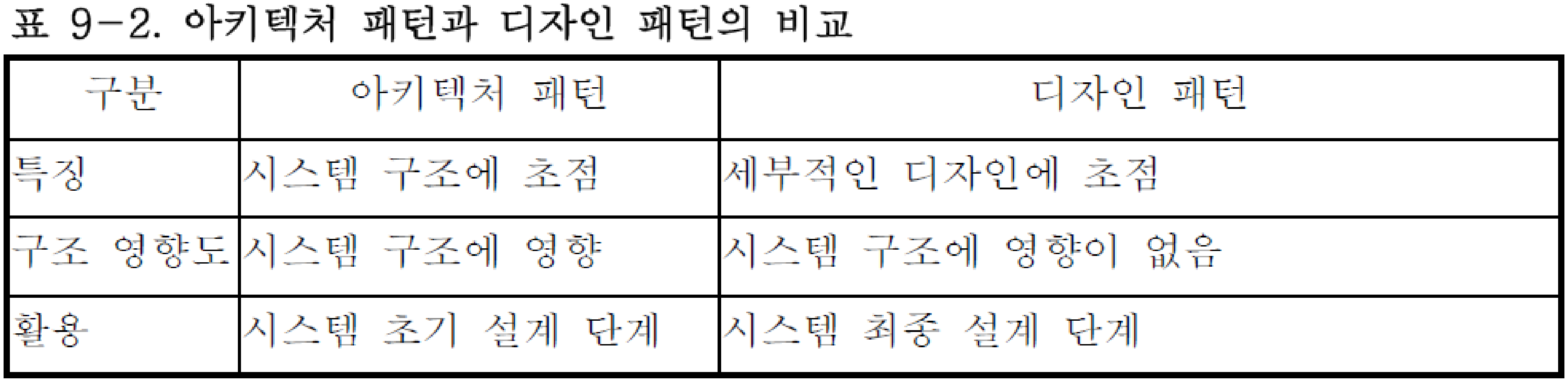


표 9-2. 아키텍처 패턴과 디자인 패턴의 비교

| 구분 | 아키텍처 패턴 | 디자인 패턴 |
|---|---|---|
| 특징 | 시스템 구조에 초점 | 세부적인 디자인에 초점 |
| 구조 영향도 | 시스템 구조에 영향 | 시스템 구조에 영향이 없음 |
| 활용 | 시스템 초기 설계 단계 | 시스템 최종 설계 단계 |

# Questions?