

Spring 2023



소프트웨어 아키텍처 패턴: Model-View-Control

Seonah Lee

Gyeongsang National University

Model-View-Control 패턴

- ▶ 패턴 정의
- ▶ 패턴 예제
- ▶ 패턴 설명
- ▶ 패턴 컴포넌트, 구조, 행위
- ▶ 패턴 구현
- ▶ 패턴 장단점

Model-View-Control Pattern: Definition

▶ 정의

- ▶ 상호작용 애플리케이션을 세 개의 컴포넌트로 분리
 - ▶ 모델(**Model**): 핵심 기능과 데이터 포함
 - ▶ 뷰(**View**): 사용자에게 정보 출력
 - ▶ 컨트롤러(**Controller**): 사용자 입력을 처리
- } 함께 사용자 인터페이스 구성
- ▶ 변경 전파 메커니즘으로 모델과 사용자 인터페이스 일관성 보장

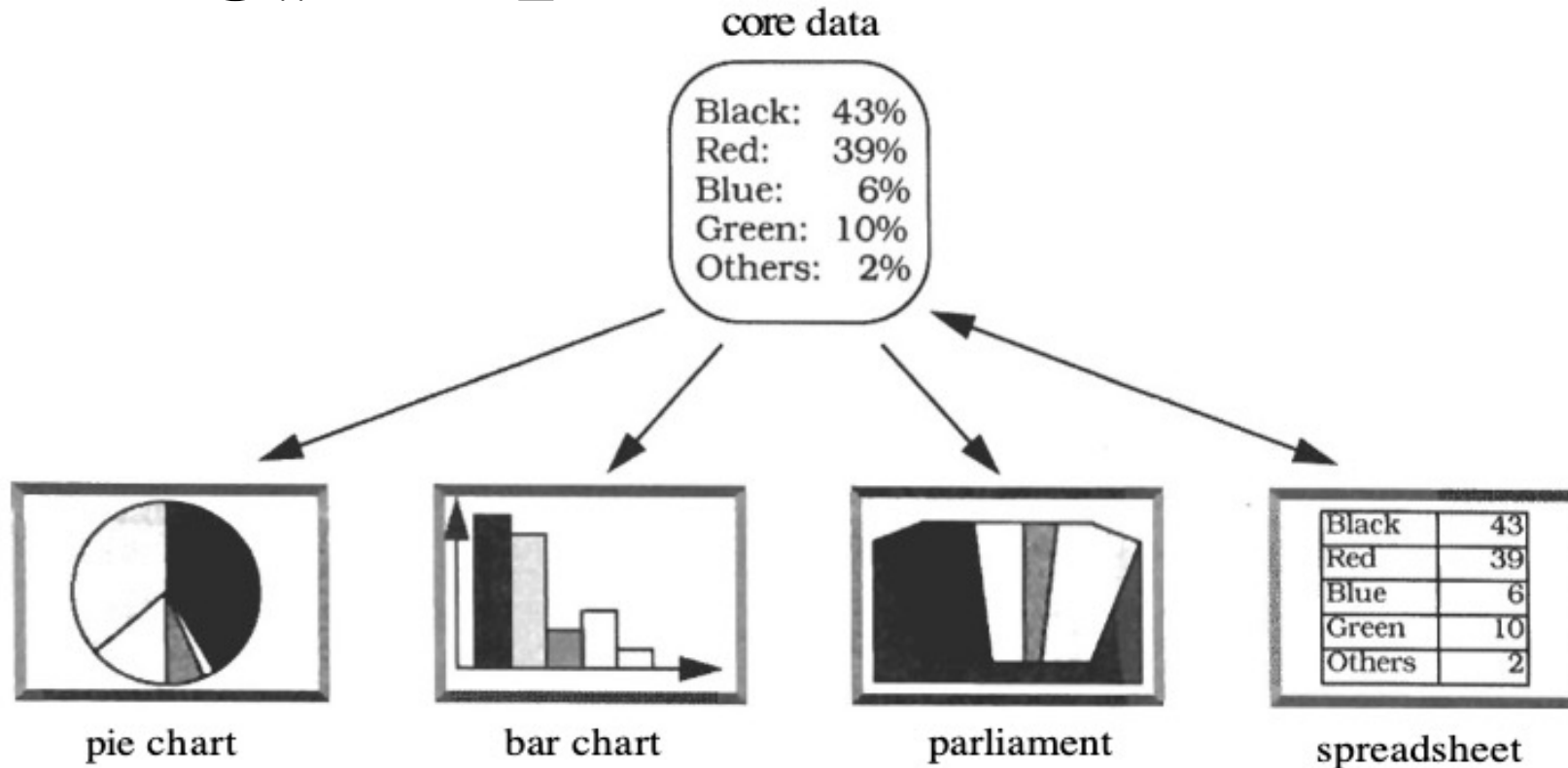
▶ 활용

- ▶ 하나의 데이터를 여러 화면으로 보일 때 활용
- ▶ 웹 기반 서비스 시스템 (거의 대부분)



Model-View-Control Pattern: Example

- ▶ 비례대표제 방식의 정당 투표 결과를 제공하는 간단한 정보 시스템
 - ▶ 데이터 입력: 스프레드시트 제공
 - ▶ 결과: 몇 가지 종류의 테이블과 차트로 표시



Model-View-Control Pattern: Description

▶ 정황(Context)

- ▶ 상호작용이 많은 시스템에 유연한 **Human-Computer Interface**를 개발하는 경우

▶ 문제(Problem)

- ▶ 사용자 인터페이스의 변경이 많이 일어나는 경우
 - ▶ 예: 애플리케이션의 기능 확장 시, 새 기능을 위한 메뉴 추가
 - ▶ 예: 다른 '룩앤필' 표준에 따르는 플랫폼에 이식될 필요가 있을 경우
- ▶ 같은 기능을 사용하는 사용자 화면이 다르게 제공해야 할 경우
 - ▶ 예: 마우스로 값 입력 **vs.** 키보드로 입력
- ▶ 유연성(**Flexibility**)를 갖춘 시스템을 구축해야 하는 경우
 - ▶ 사용자 인터페이스와 핵심 기능이 얹혀 있으면, 비용이 많이 들고 오류 발생 확률이 높아짐
 - ▶ 사용자 인터페이스들을 따로 따로 개발하면, 향후 변경에 수 많은 모듈을 일일이 검토해야 함

Model-View-Control Pattern: Description

▶ 해법(Solution)

- ▶ 상호작용 애플리케이션을 **Model, View, Controller** 이렇게 3개의 영역으로 구분
- ▶ **Model** : 핵심 데이터와 기능을 캡슐화
 - ▶ 특정 출력 방식이나 입력 동작에 영향을 받지 않고 독립적임
- ▶ **View** : 사용자의 정보를 화면에 표시
 - ▶ 모델로부터 데이터를 얻음
 - ▶ 각 뷰마다 각 **Controller**가 하나씩 연결됨
- ▶ **Controller** : **View**로부터 입력 정보를 받아 관련된 **View**나 **Model**을 호출함
 - ▶ 키보드 입력, 마우스 이동, 마우스 버튼의 동작 등을 이벤트 형태로 입력 받음; 요청으로 변환

Model-View-Control Pattern: Components

Model

- 애플리케이션의 핵심 기능을 제공
- 관련된 뷰와 컨트롤러를 등록
- 데이터가 변경된 사실을 종속된 컴포넌트들에 통지

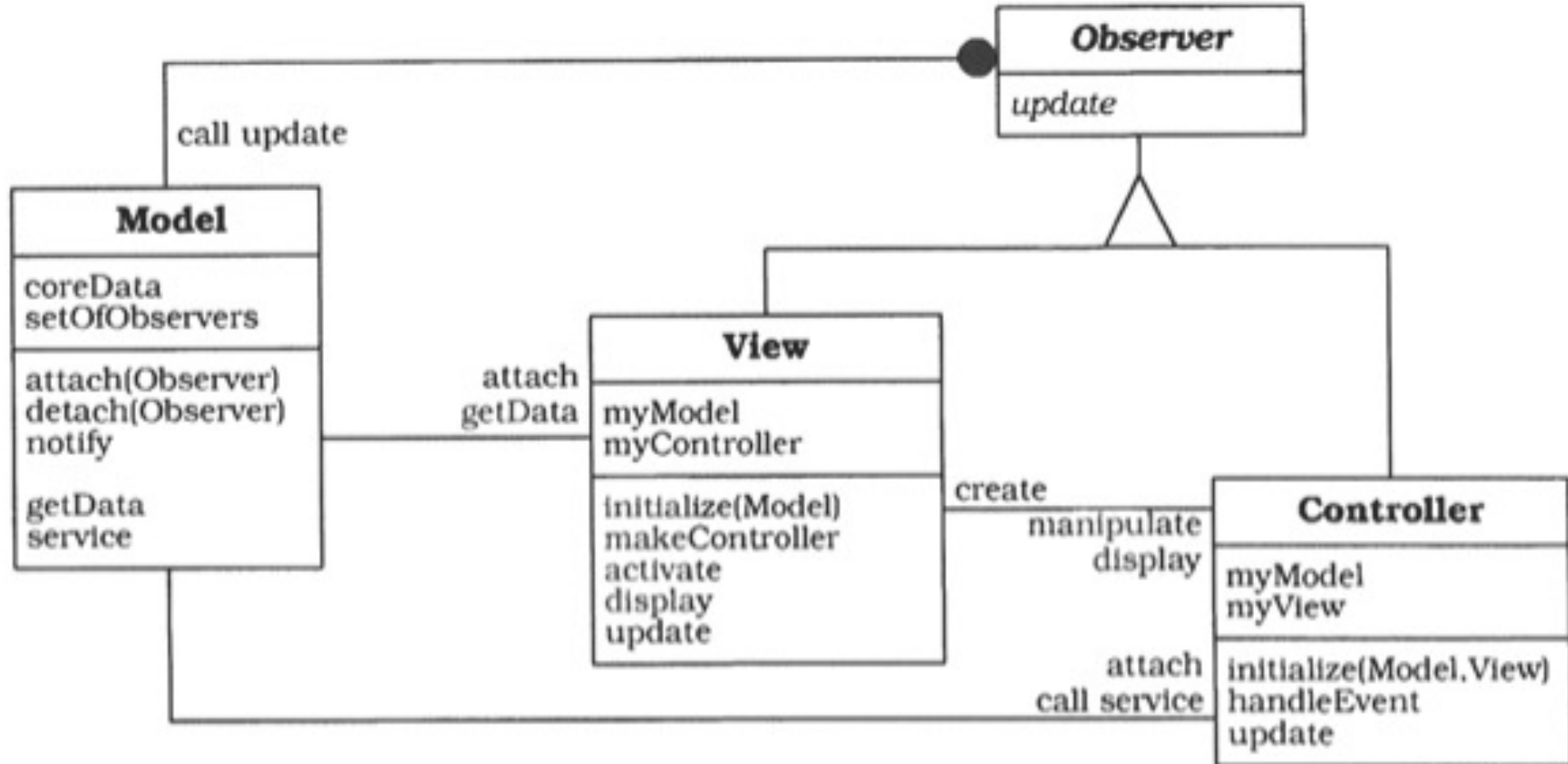
View

- 관련된 컨트롤러를 생성하고 초기화
- 업데이트 프로시저를 구현
- 모델로부터 데이터를 가져옴

Controller

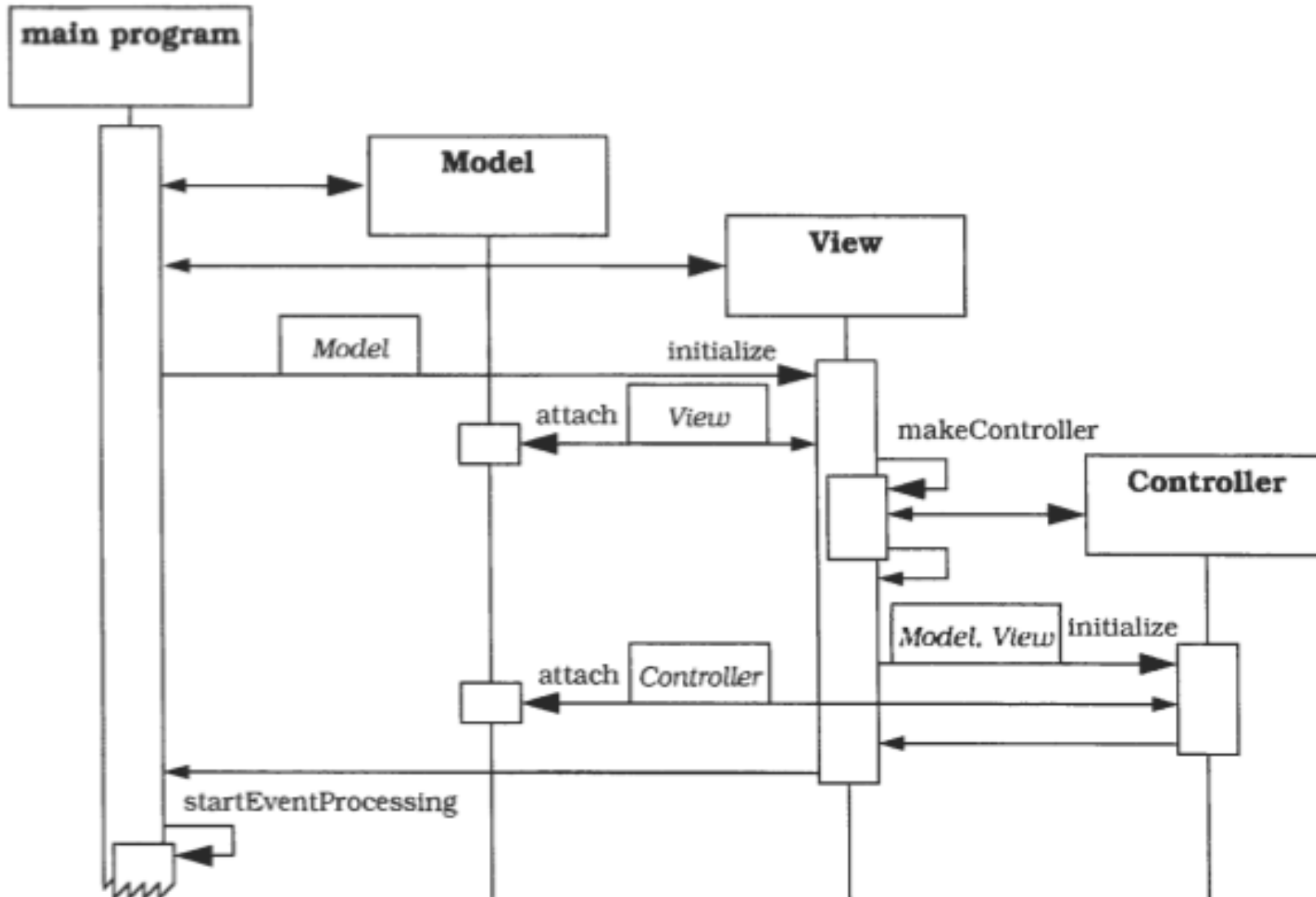
- 사용자 입력을 이벤트 형태로 받음
- 이벤트를 모델에 대한 서비스 요청이나 뷰에 대한 디스플레이 요청으로 변환
- 필요하다면 업데이트 프로시저를 구현

Model-View-Control Pattern: Structure



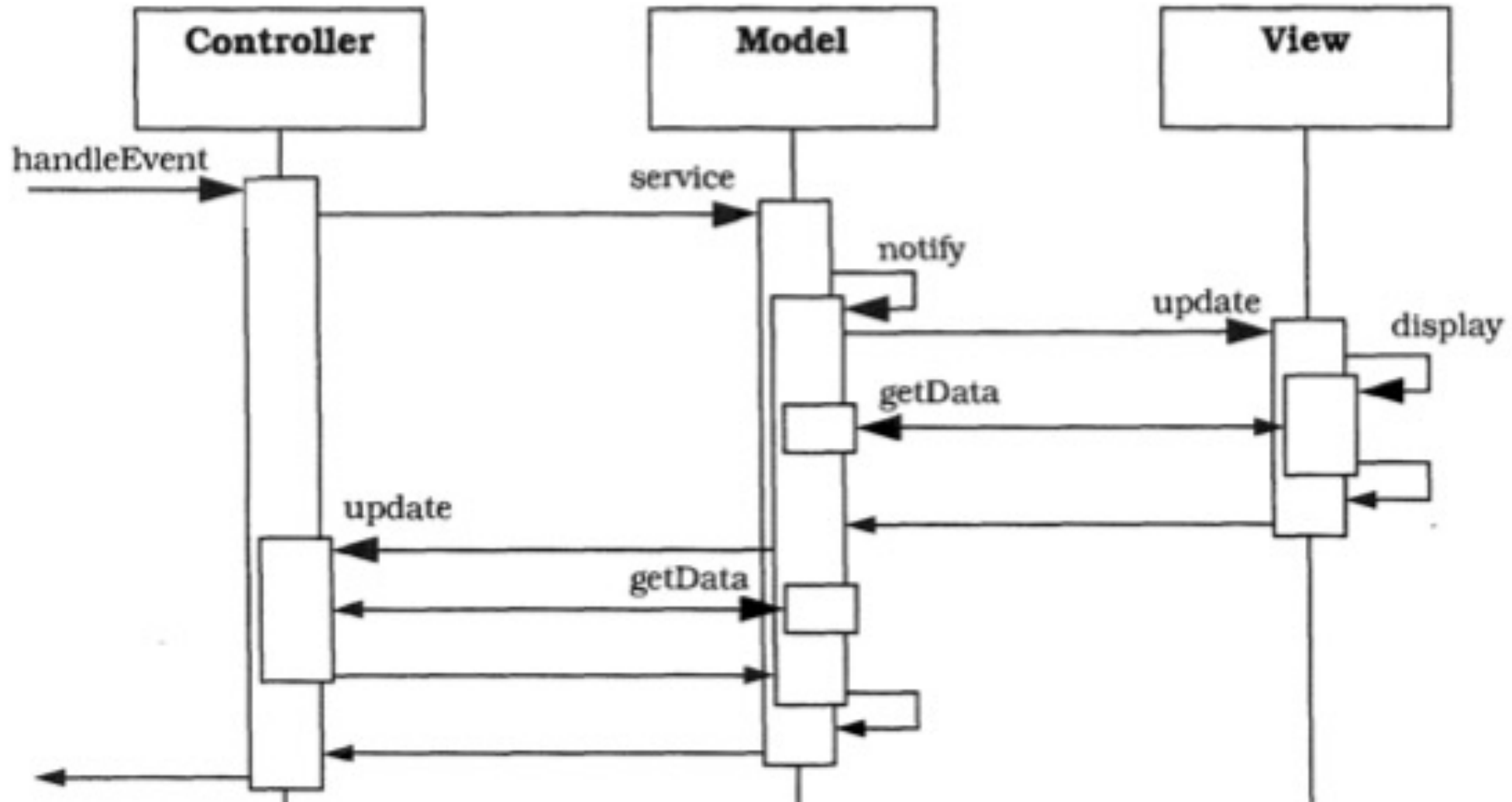
Model-View-Control Pattern: Behavior

▶ 초기화



Model-View-Control Pattern: Behavior

- ▶ 사용자 입력에 의한 변경 과정



Model-View-Control Pattern: Realization

▶ 구현 순서

1. 서비스의 핵심 기능과 사용자 인터페이스 부분을 분리

- ▶ 입력 동작과 출력 동작으로 부터 핵심 기능을 분리한다.
- ▶ 핵심적인 데이터와 기능을 캡슐화하기 위한 모델 컴포넌트를 설계한다.
- ▶ 뷰와 컨트롤러를 위해 필요한 인터페이스를 고려하여 모델에 추가한다.

2. Observer 패턴을 적용하여 모델을 구현

- ▶ 상기 패턴을 활용하여 모델에 **Publisher**의 역할을 부여한다.
- ▶ 뷰와 컨트롤러를 **Subscriber**로 등록하거나 등록 해제하는 프로시저를 모델에 구현한다.
- ▶ 통지 프로시저를 구현한다.

Model-View-Control Pattern: Realization

▶ 구현 순서

3. View를 설계하고 구현

- ▶ 각 뷰의 외양(**appearance**)을 설계한다.
- ▶ 화면에 뷰를 보이기 위한 프로시저를 정의하고 구현한다.
 - ▶ 이 프로시저는 디스플레이 할 데이터를 모델로부터 얻어온다.
- ▶ 모델에서 일어난 변경을 반영하기 위해 업데이트 프로시저를 구현한다.

4. Controller를 설계하고 구현

- ▶ 각 뷰마다 사용자 동작에 응답하는 시스템 동작을 정의한다.

5. View와 Controller 관계를 설계하고 구현

- ▶ 대체적으로 뷰는 스스로 초기화하는 동안 자신과 관련된 컨트롤러를 생성한다.

Model-View-Control Pattern: Realization

▶ 구현 순서

6. MVC의 설정 코드를 구현

- ▶ 모델을 초기화한 후, 뷰를 생성하고 초기화한다. 이후 이벤트 프로세싱을 시작한다.

7. 동적 뷰를 생성

- ▶ 애플리케이션이 동적으로 뷰를 열고 닫는 기능을 허용 시, 열린 뷰를 관리하는 컴포넌트를 제공한다.

8. 컨트롤러를 플러그처럼 장착할 수 있도록 구현

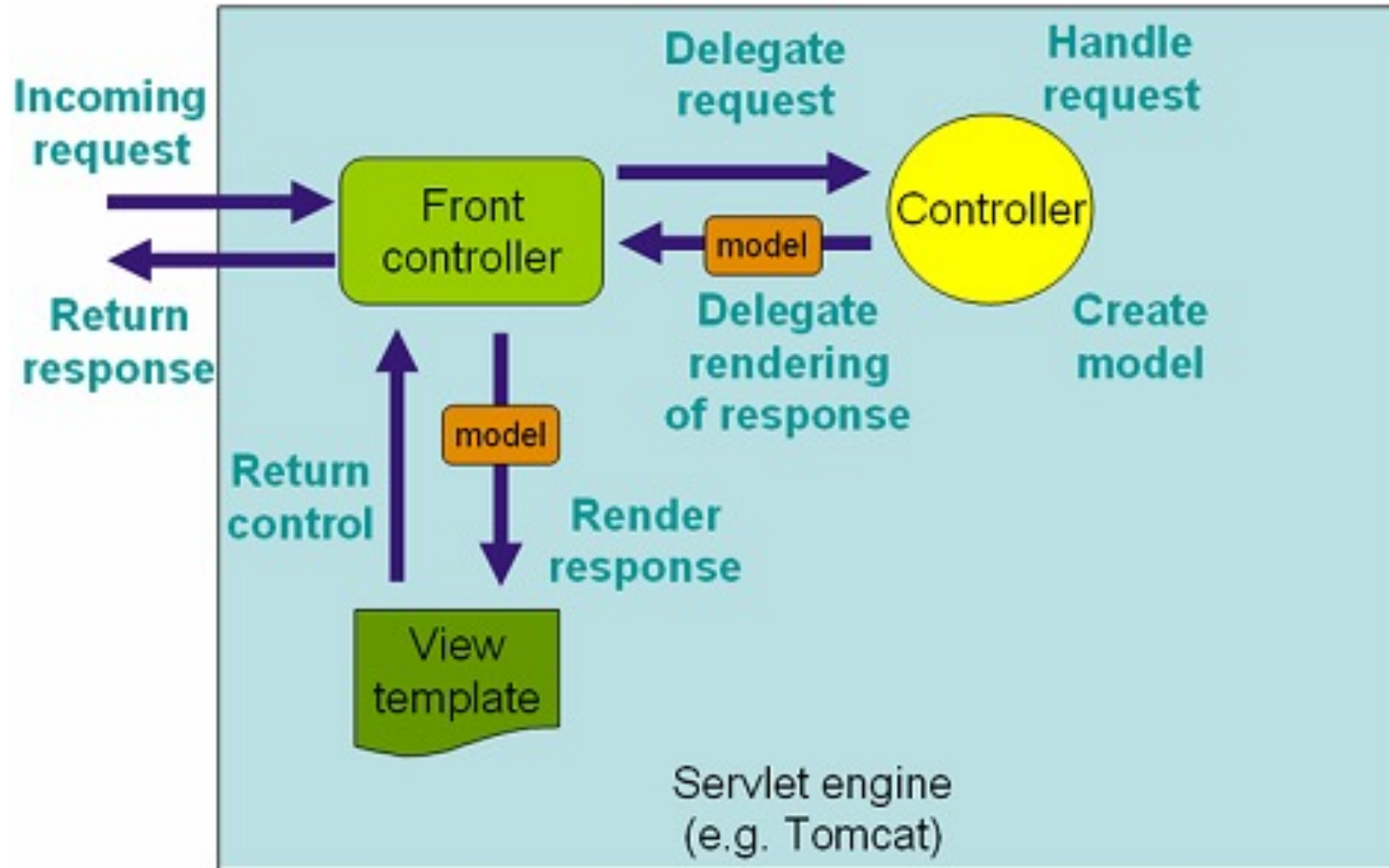
- ▶ 하나의 뷰에 여러 개의 각기 다른 컨트롤러를 조합할 수 있도록 제어 측면을 분리하면 좋다.

9. 계층적인 뷰와 컨트롤러를 위한 인프라를 구축

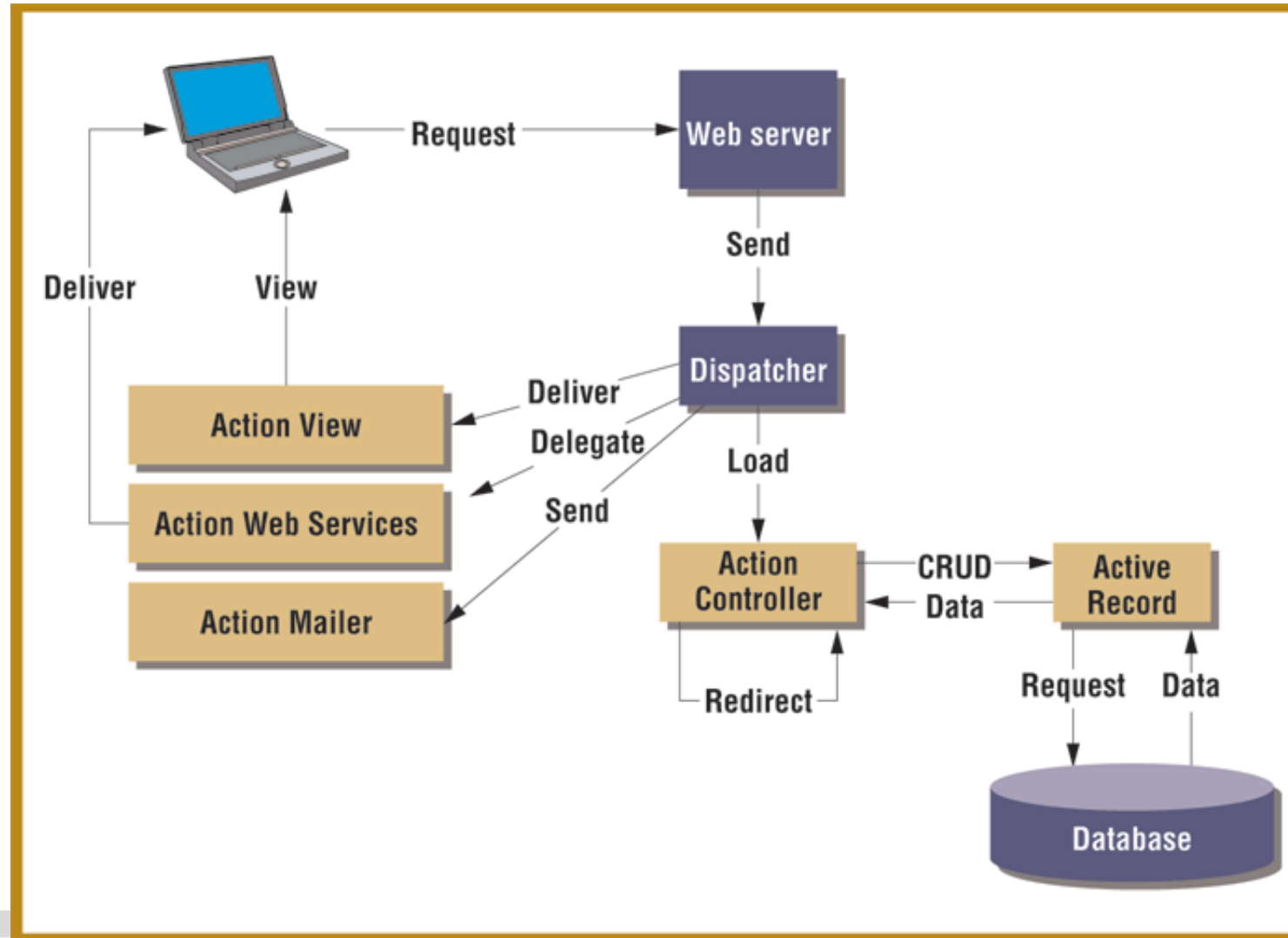
- ▶ 사용자 인터페이스를 미리 정의된 뷰 객체들을 조합하여 구축할 때 고려할 수 있다.

10. 결합도를 낮춰 시스템 종속성을 줄여감

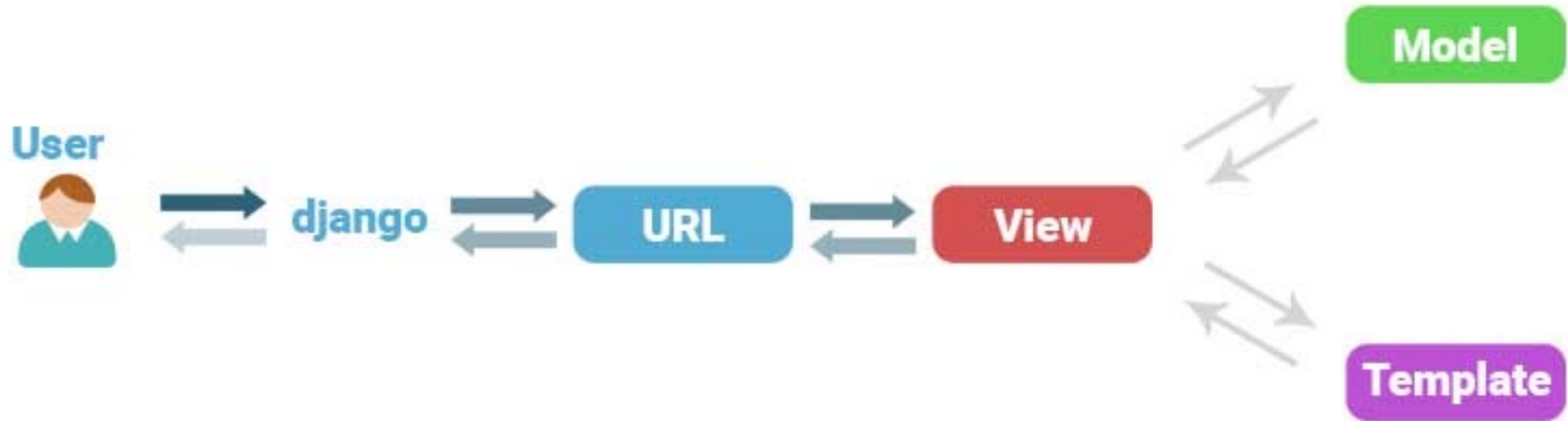
Model-View-Control Pattern: Case Studies



Model-View-Control Pattern: Case Studies



Model-View-Control Pattern: Case Studies



Model-View-Control Pattern: Benefits

- ▶ 동일한 모델로부터 여러 **View** 들을 표현할 수 있음
 - ▶ 모델과 사용자 인터페이스 컴포넌트를 엄격히 분리, 다중 뷰 구현 가능
- ▶ **View**들을 동기화할 수 있음
 - ▶ 모델의 변경전파 메커니즘에 의하여 데이터 변경사항 통지
- ▶ **View**의 추가, 변경, 삭제가 자유로움
 - ▶ **MVC** 애플리케이션을 새로운 플랫폼에 이식할 때 핵심 기능 영향 없음
- ▶ 프레임워크로 확장하여 구현이 가능함

Model-View-Control Pattern: Liabilities

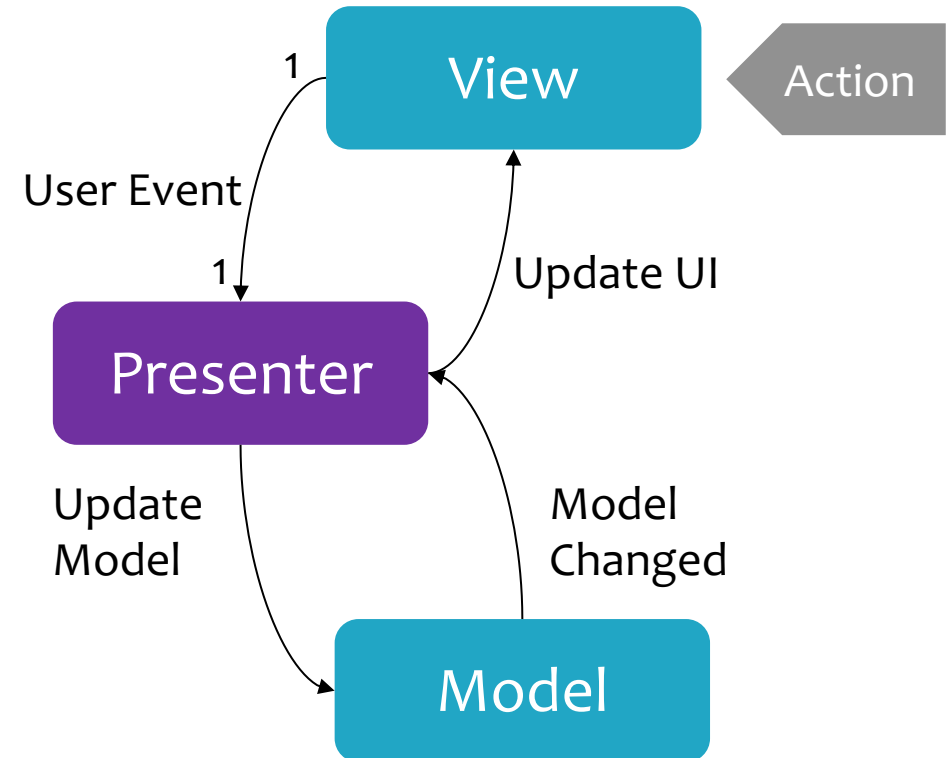
- ▶ 설계가 복잡하고 개발이 어려움
 - ▶ 단순한 텍스트 요소들을 위해서 **MVC** 사용 시 복잡성만 증가
 - ▶ 하나의 사용자 동작으로 많은 업데이트가 발생할 수 있음
 - ▶ 불필요한 변경 통지들에 대해서 무시하는 메커니즘 강구 필요
- ▶ **View와 Controller**는 밀접히 관련되어 있음
 - ▶ 제각기 독립적으로 재사용하기 힘들 수 있음
- ▶ **View와 Controller**는 모델에 의존
 - ▶ 모델의 인터페이스를 변경하면 **View와 Controller**로 변경 되어야 함

Ref: Model-View-Presenter

▶ Three inter-connected parts

- ▶ **Model:** encapsulates of the underlying data
- ▶ **View:** provides an interface for user input and displays information to the user
- ▶ **Presenter:** acts like mediator between the model and the view.
 - ▶ It receives data from the model, formats it and deliver to the view

▶ Remove dependency between model and view

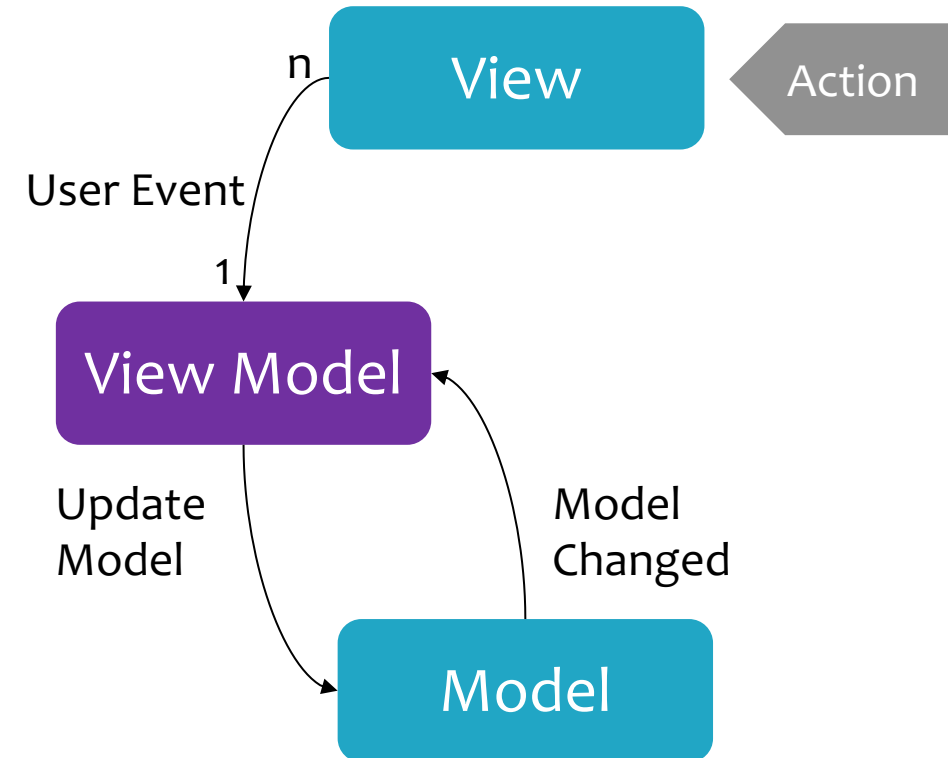


View와 Presenter의 의존성 강함

Ref: Model-View-View Model

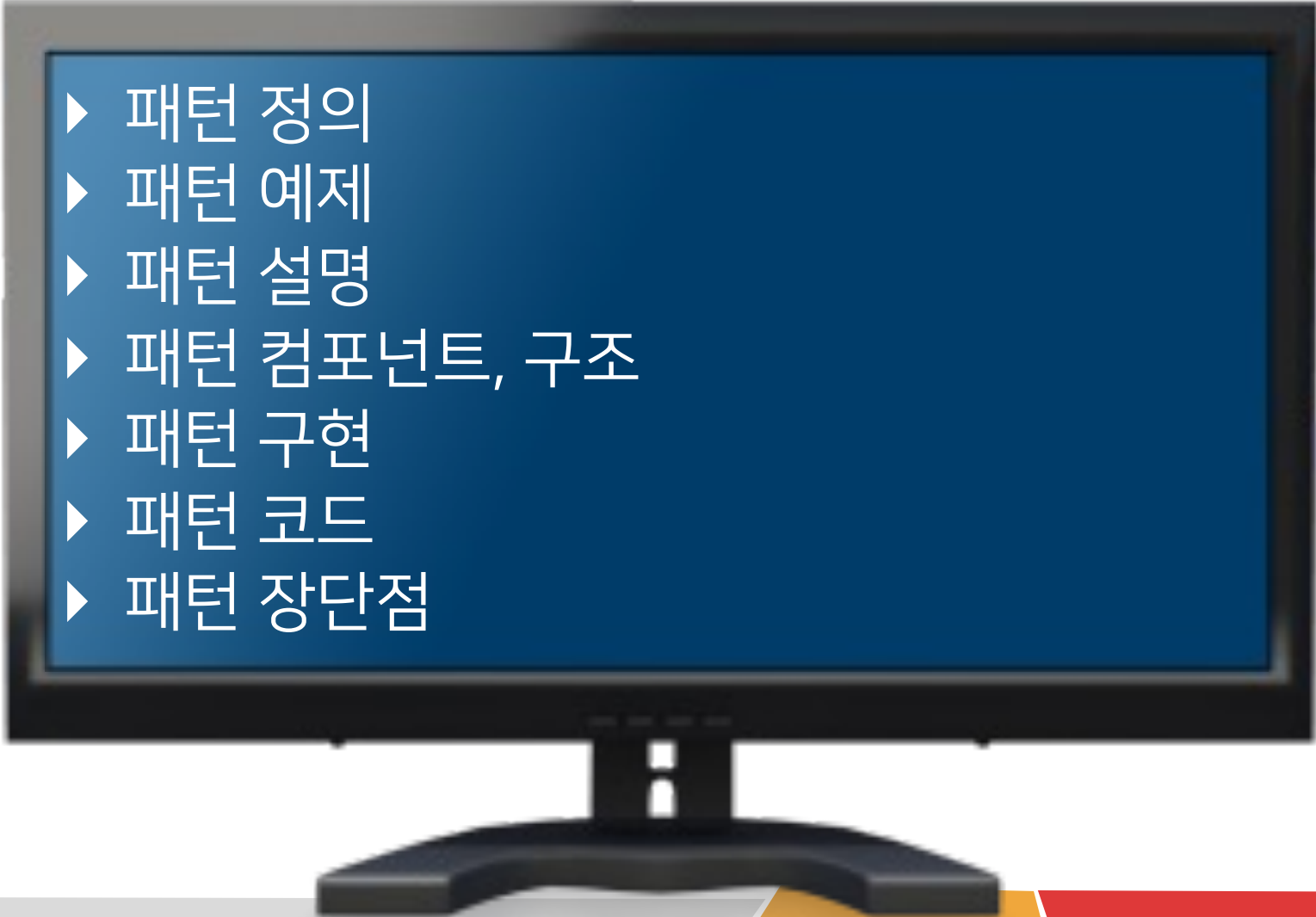
▶ Three inter-connected parts

- ▶ **Model:** This component represents the data and business logic of the application.
- ▶ **View:** This component represents the UI of the application.
- ▶ **ViewModel:** This component acts as a bridge between the view and the model.
 - ▶ It exposes data from the model to the view and provides methods and commands
 - ▶ The view can use the view model to manipulate the data in the model.



각각이 독립적; 설계는 쉽지 않음

○○ Presentation-Abstraction-Control 패턴 ○○

- 
- ▶ 패턴 정의
 - ▶ 패턴 예제
 - ▶ 패턴 설명
 - ▶ 패턴 컴포넌트, 구조
 - ▶ 패턴 구현
 - ▶ 패턴 코드
 - ▶ 패턴 장단점

Presentation-Abstraction-Control Pattern: Definition

▶ 정의

- ▶ 특정 기능을 가진 에이전트들이 추상화 여부에 따라 계층 구조를 이루며 협력하는 패턴
- ▶ 에이전트는 **Presentation, Abstraction, Control** 3개의 컴포넌트로 구성
 - ▶ **Abstraction** 컴포넌트는 **MVC** 패턴의 **Model**에 해당
 - ▶ **Presentation** 컴포넌트는 **MVC** 패턴의 **View, Controller**에 해당
 - ▶ **Control** 컴포넌트는 **MVC** 패턴에 해당하는 컴포넌트가 없음

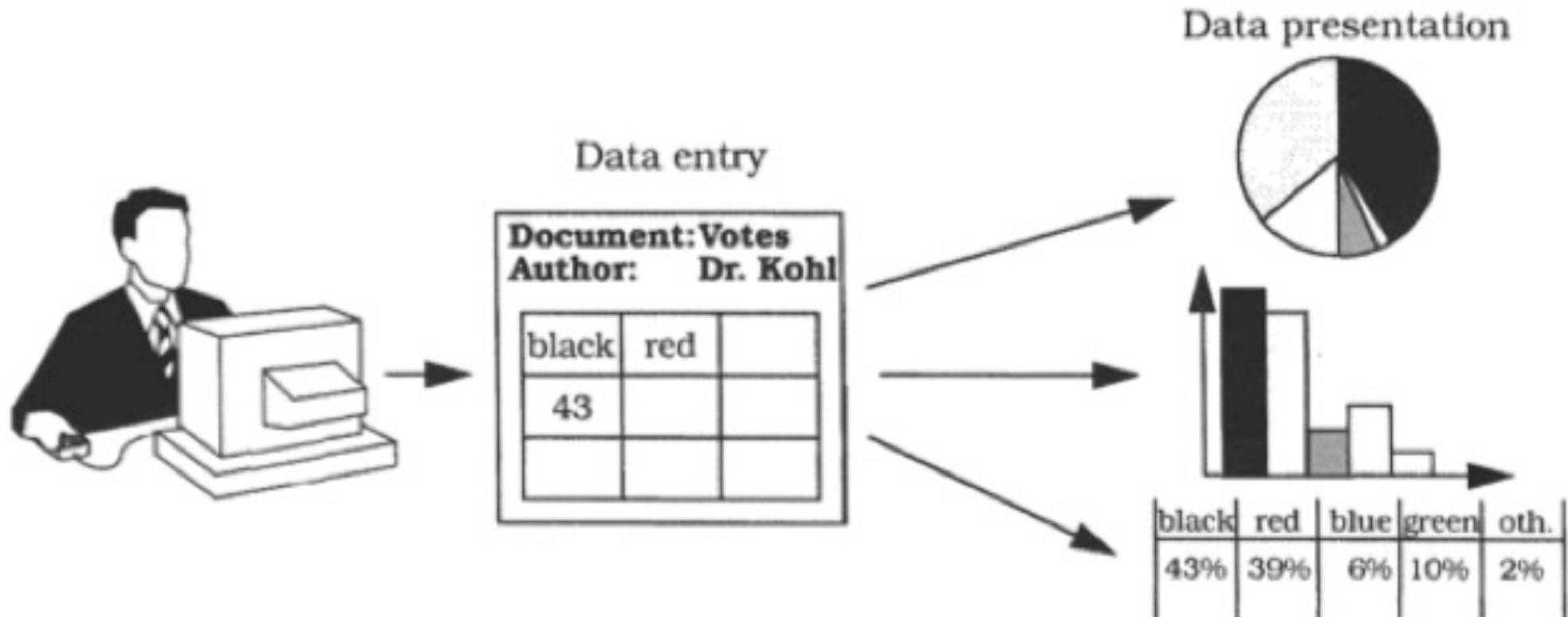
▶ 활용

- ▶ 네트워크 트래픽 관리, 모바일 로봇

Presentation-Abstraction-Control Pattern: Example

▶ 예제

- ▶ 비례대표제 방식의 정당 투표 결과를 제공하는 정보 시스템



Presentation-Abstraction-Control Pattern: Description

▶ 정황(Context)

- ▶ **Agent** 들이 협력하는 상호작용이 많은 애플리케이션
 - ▶ **Agent-1** : HCI 담당(사용자 입력 및 데이터 출력 담당)
 - ▶ **Agent-2** : 특정 데이터 모델과 이와 관련된 기능을 담당
 - ▶ **Agent-3** : 에러 핸들링 타 시스템과 연동
 - ▶

▶ 문제(Problem)

- ▶ 에이전트들은 자체 상태와 데이터를 저장하고 있지만 서로 협력해야 함.
- ▶ 에이전트들 간의 교환, 데이터, 메시지, 이벤트를 위한 단일 메커니즘을 만들어야 함
- ▶ 개별 에이전트의 변경, 추가, 삭제로 인한 전체 시스템의 변화가 일어나서는 안됨

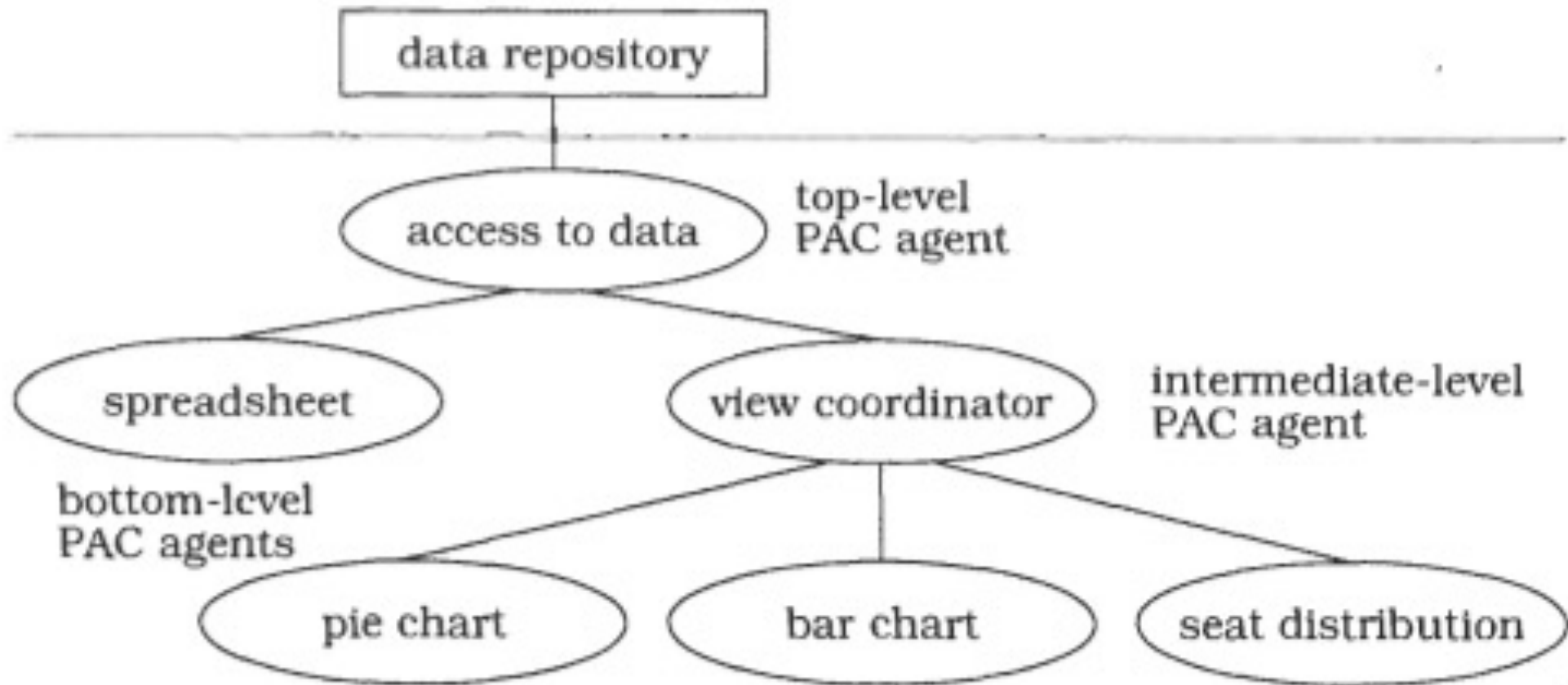
Presentation-Abstraction-Control Pattern: Description

▶ 해법(Solution)

- ▶ 다음과 같은 계층 구조로 **PAC** 에이전트를 만듦
 - ▶ 최상위 레벨 **PAC** 에이전트 : 시스템의 핵심 기능을 제공(예. 메뉴 바, 대화상자 등)
 - ▶ 중간 레벨 **PAC** 에이전트 : 상위, 하위를 연결하여 조합하는 기능을 제공
 - ▶ 예. 건축의 외부 입면도, 평면도 등
 - ▶ 하위 레벨 **PAC** 에이전트 : 사용자가 다루는 자체 독립적 의미 개념 제공
 - ▶ 예. 스트레드시트, 차트 등
- ▶ 에이전트는 **Presentation-Abstraction-Control** 컴포넌트로 구성
 - ▶ **Presentation** : 시각적 동작을 제공
 - ▶ **Abstraction** : 에이전트의 기반을 이루는 데이터 모델과 이를 처리하는 기능 제공
 - ▶ **Control** : **Presentation**과 **Abstraction**을 연결하며 특정 에이전트가 다른 에이전트와 통신할 수 있는 기능 제공

Presentation-Abstraction-Control Pattern: Description

- ▶ 비례대표제 방식의 정당 투표 결과를 제공하는 정보 시스템의 **PAC** 패턴



Presentation-Abstraction-Control Pattern: Components

Top-level Agent

- 시스템의 핵심 기능을 제공
- PAC 계층 구조를 제어

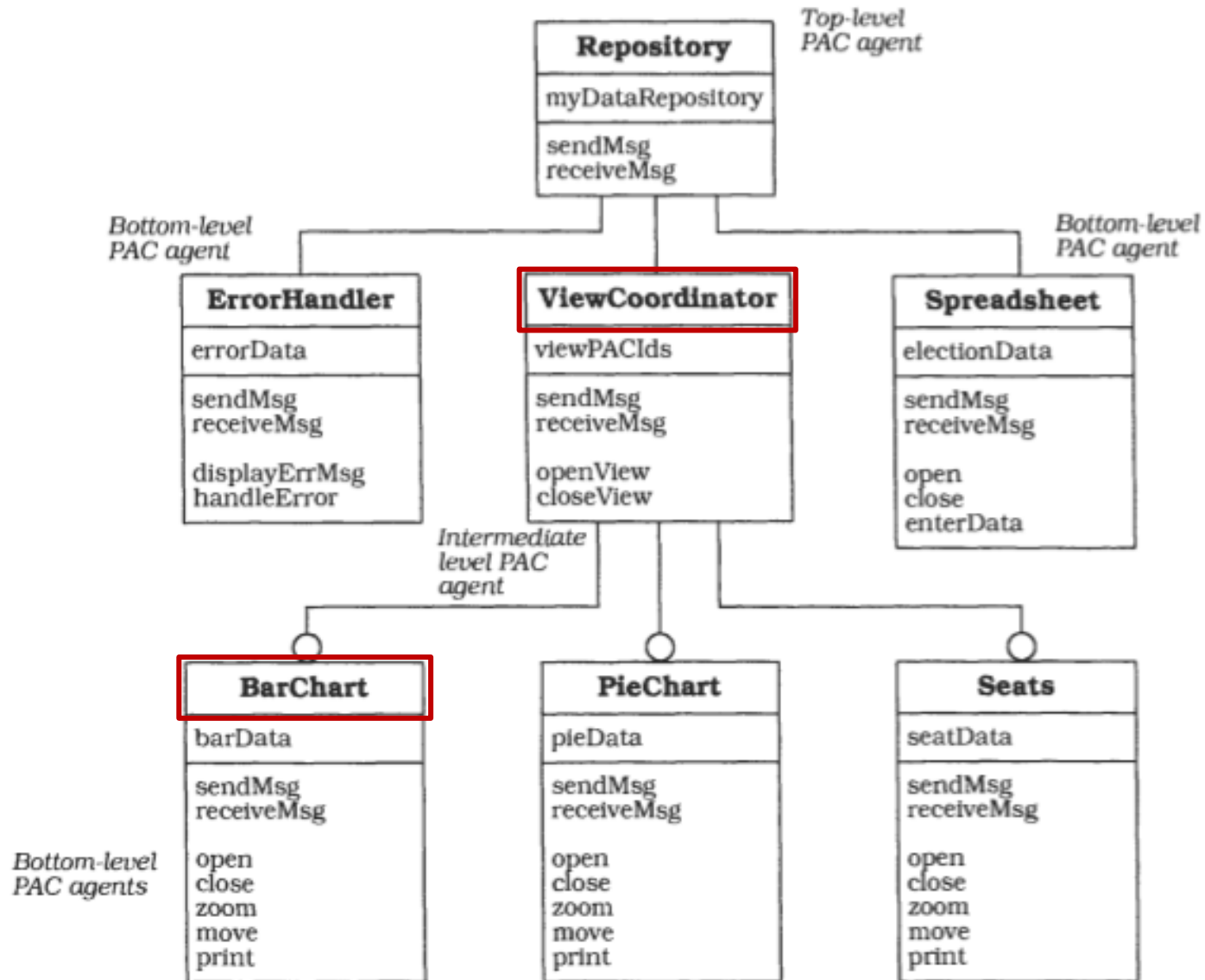
Intermediate-level Agent

- 하위 레벨 PAC 에이전트들을 조정
- 하위 레벨 PAC 에이전트들을 하나의 상위 추상 단위로 조합

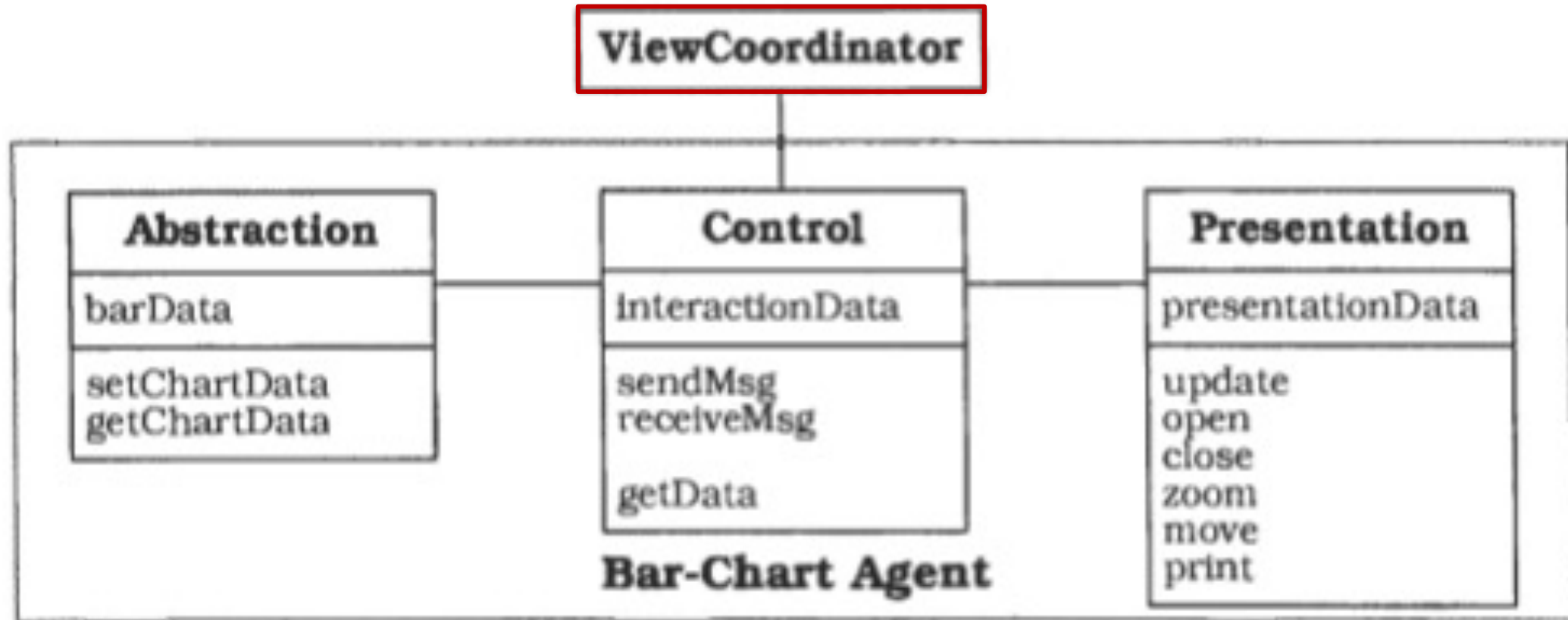
Bottom-level Agent

- 소프트웨어의 특정 뷰나 시스템 서비스를 제공 (이 때 필요한 사람-컴퓨터 상호작용 측면도 함께 제공)

Presentation-Abstraction-Control Pattern: Structure

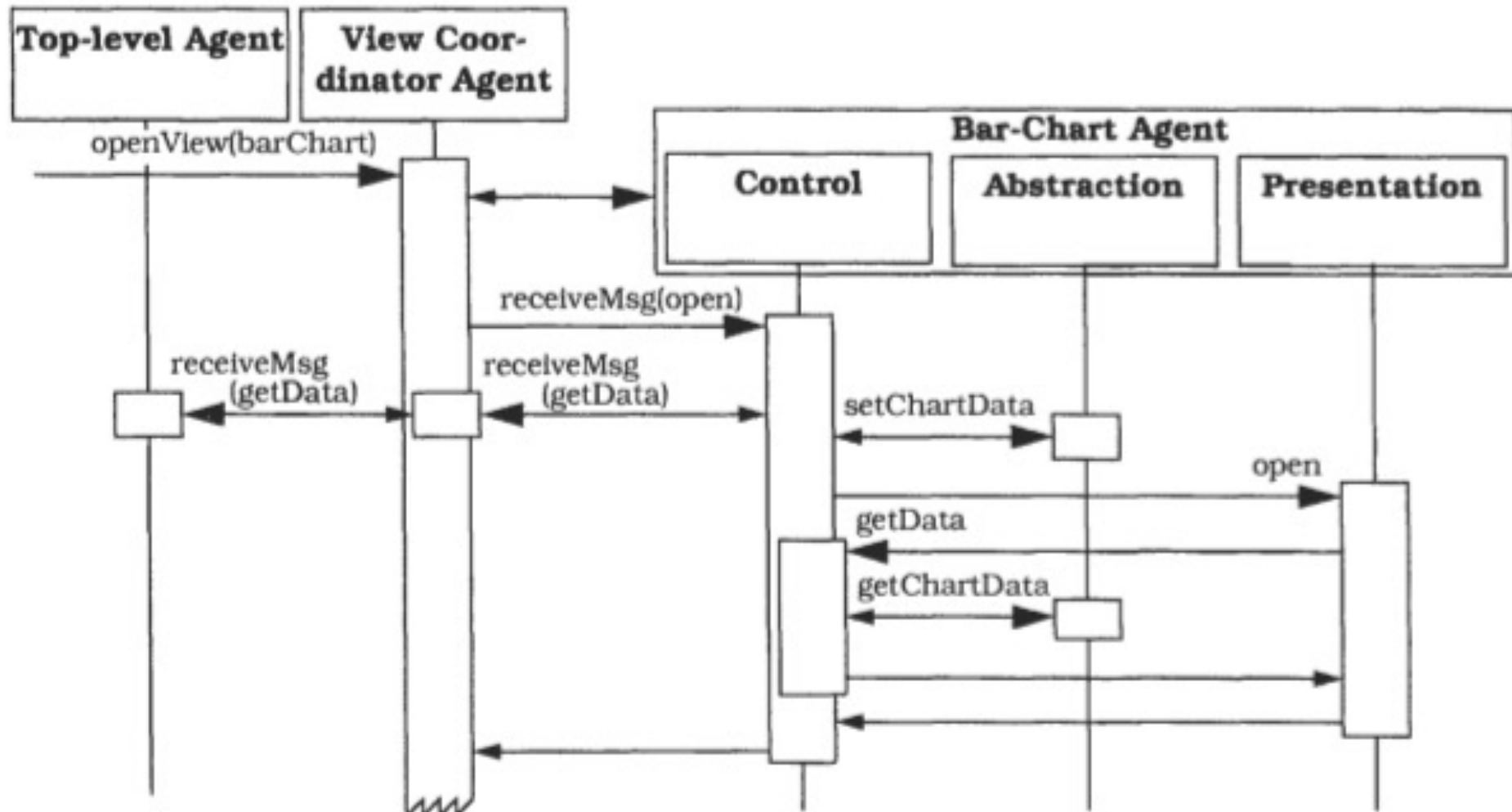


Presentation-Abstraction-Control Pattern: Structure



Presentation-Abstraction-Control Pattern: Behavior

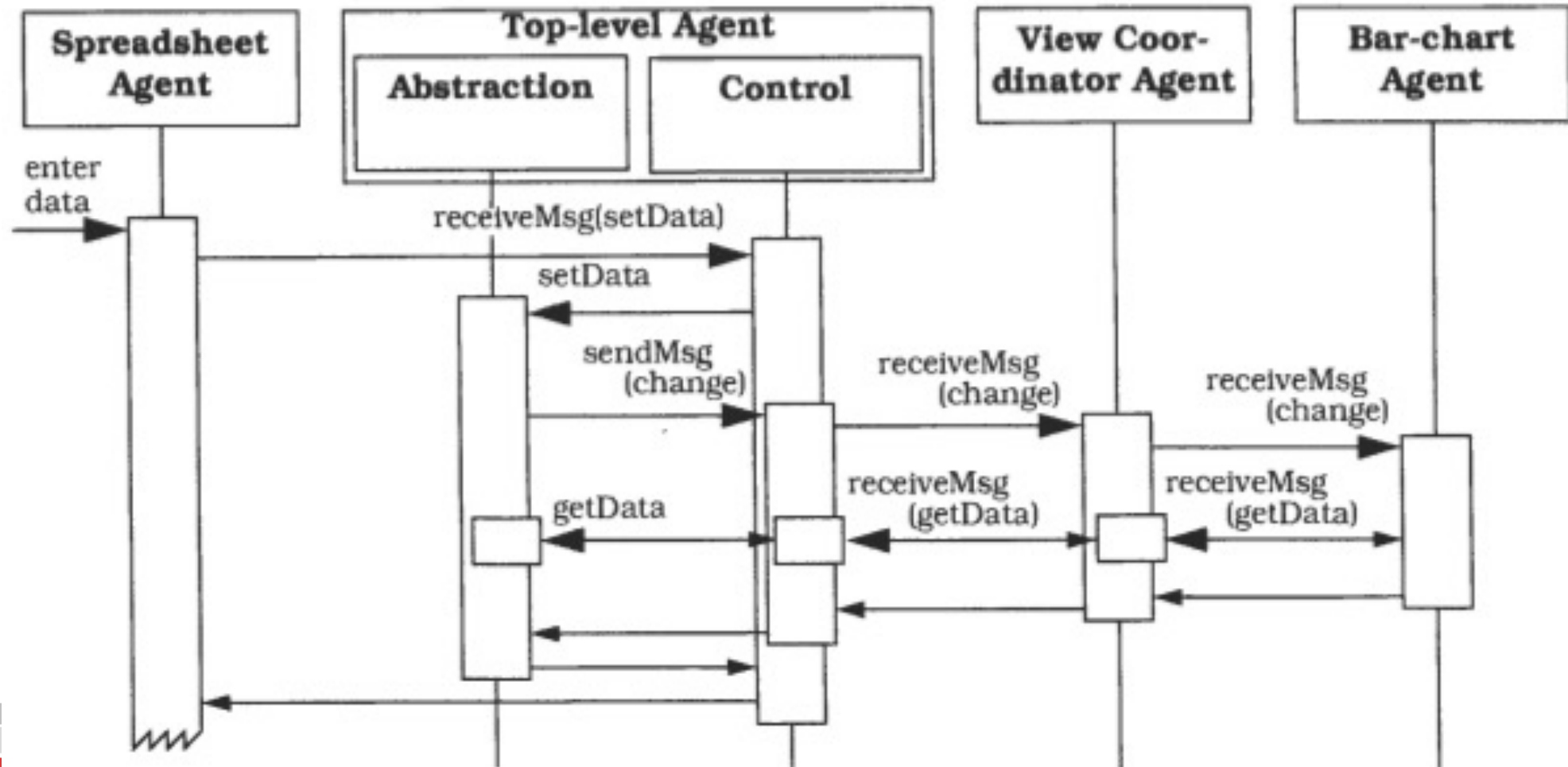
- ▶ 데이터의 바 차트를 열 때 에이전트의 협력



Presentation-Abstraction-Control

Pattern: Behavior

- ▶ 새 데이터를 입력한 후의 에이전트의 동작



Presentation-Abstraction-Control Pattern: Realization

▶ 구현 순서

1. 애플리케이션 모델을 정의. 도메인을 적절하게 분해하고 조직화하는 방법을 찾음
 - ▶ 시스템이 제공할 서비스, 컴포넌트 책임, 컴포넌트 관계 및 협력, 컴포넌트의 데이터 처리 및 상호작용 고려
2. **PAC** 계층 구조화를 위한 전략을 정의
 - ▶ 협력 에이전트의 계층 구조를 조직화하기 위한 일반적인 가이드라인을 정의할 수 있음
3. 최상위 레벨 **PAC** 에이전트를 정의
 - ▶ 시스템의 핵심 기능에 해당하는 분석 모델의 부분들; 전역 데이터 모델/인터페이스를 위한 컴포넌트들
4. 사용자 조작이나 데이터 출력과 같이 독립적으로 돌아가는 최하위 레벨 **PAC** 에이전트들을 정의
 - ▶ 자체 주요 목적과 직접적으로 관련되지 않은 부가적인 서비스를 포함
5. 오류 처리 등의 시스템 서비스를 제공하는 최하위 레벨 **PAC** 에이전트들을 정의

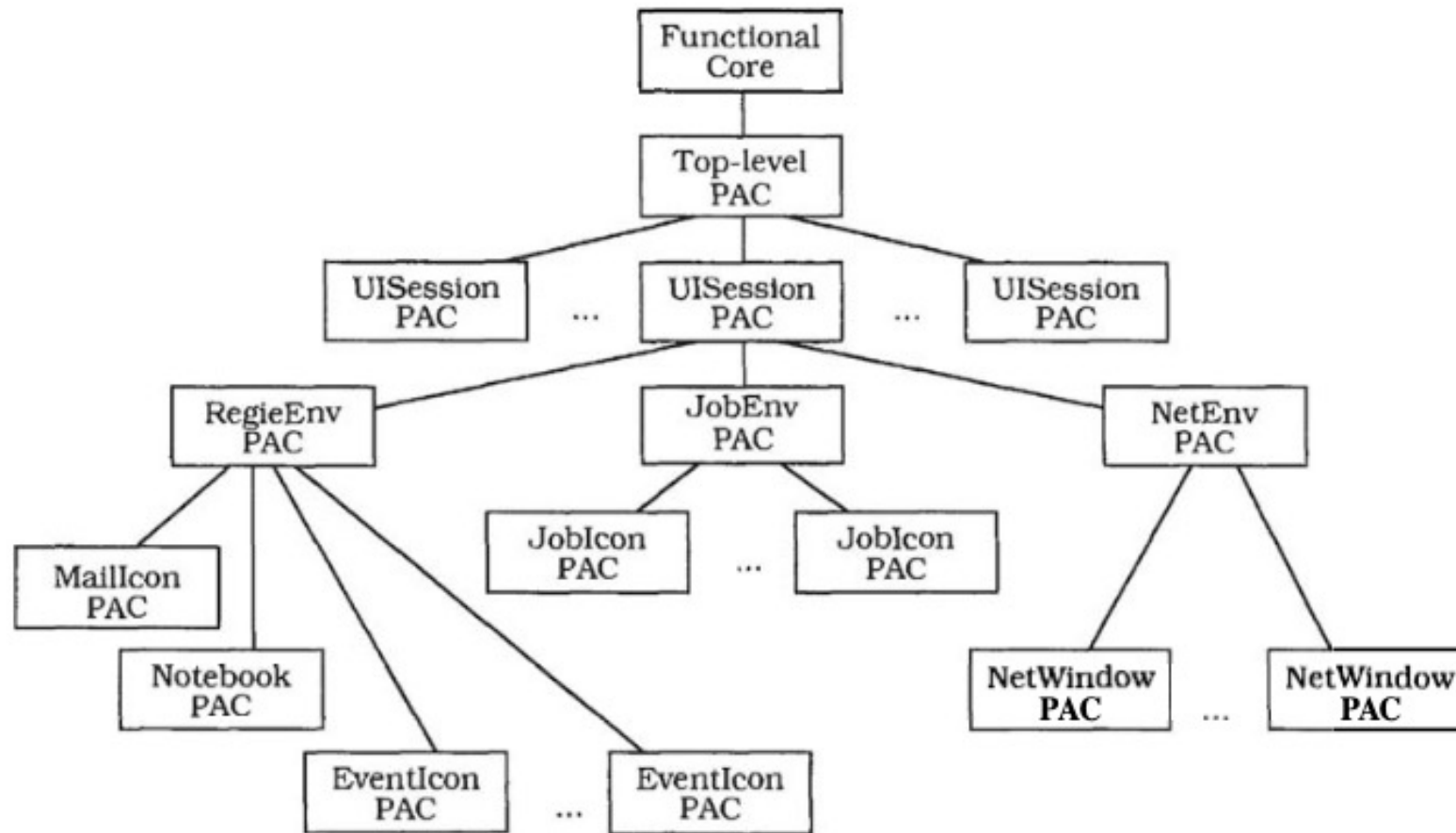
Presentation-Abstraction-Control Pattern: Realization

▶ 구현 순서

6. 하위 레벨 **PAC** 에이전트들을 조합하기 위한 중간 레벨 **PAC** 에이전트들을 정의
 - ▶ 화면 표시, 스크롤, 줌 등의 동작을 개별 에이전트로 구현할 시, 자신을 구성하는 하위 에이전트들을 조작
7. 하위 레벨 **PAC** 에이전트들을 조정하기 위한 중간 레벨 **PAC** 에이전트들을 정의
 - ▶ 동일한 의미적 개념에 대해 여러 뷰를 제공하는데, 이러한 뷰들의 코디네이터 역할 등을 맡을 수 있음
8. **HCI**에서 핵심 기능을 분리
 - ▶ 모든 **PAC** 에이전트마다 프리젠테이션 컴포넌트와 추상 컴포넌트를 도입
9. 외부 인터페이스를 설계
 - ▶ **PAC** 에이전트가 다른 에이전트와 협력, 이벤트와 데이터를 교환하는 기능을 제어 컴포넌트에 구현한
10. 계층 구조로 연결
 - ▶ 개별 **PAC** 에이전트를 구현한 후 최종 **PAC** 계층 구조를 구축

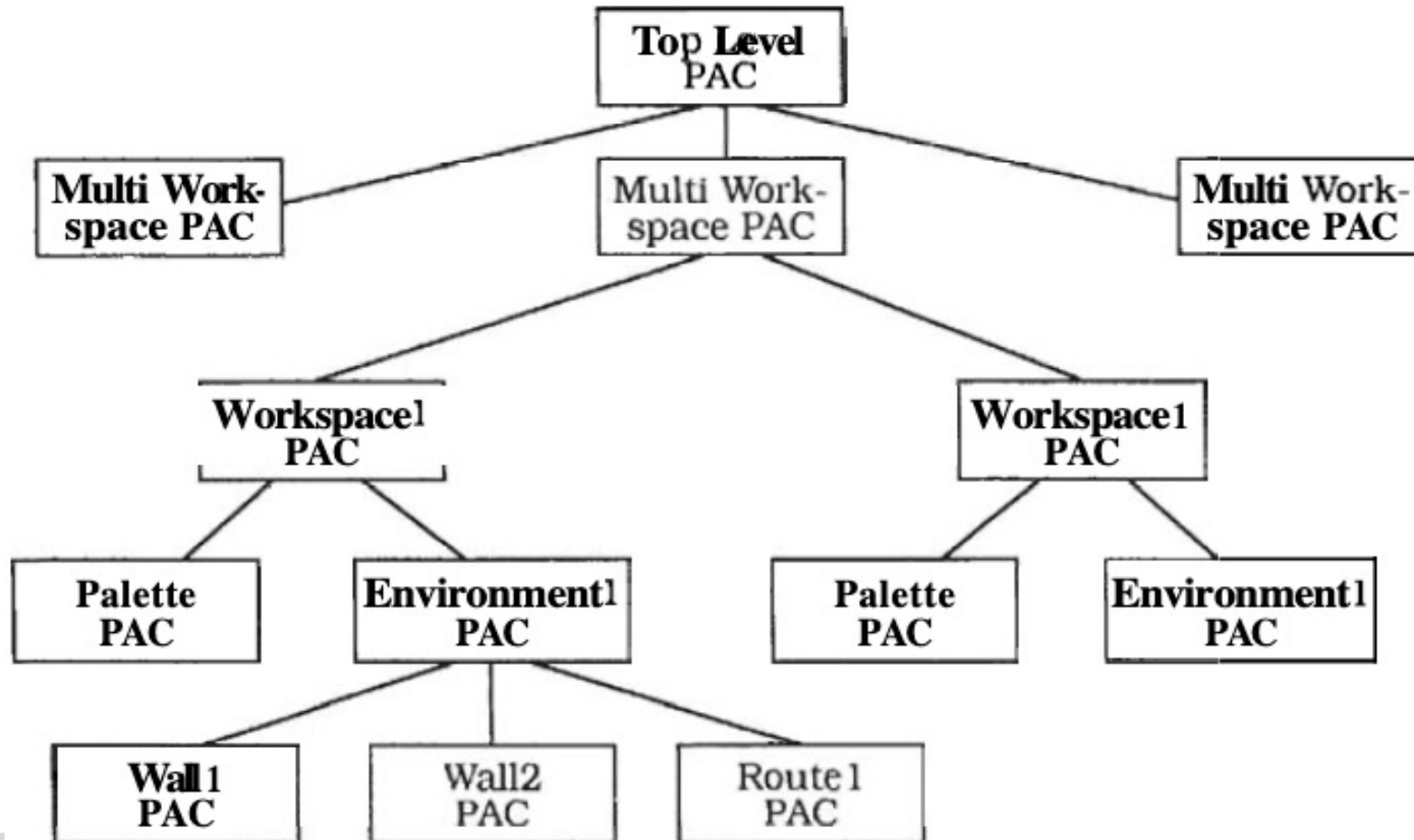
Presentation-Abstraction-Control Pattern: Case Studies

► Network Traffic Management



Presentation-Abstraction-Control Pattern: Case Studies

► Mobile Robot



Presentation-Abstraction-Control Pattern: Benefits

- ▶ 관심사항(**Concerns**)을 명확히 분리
 - ▶ 애플리케이션 도메인에서 각기 다른 의미적 개념이 독립적인 에이전트로 표현
 - ▶ 각 에이전트는 자체 상태와 데이터를 저장, 독립성 유지
- ▶ 가변성과 확장성을 지원
 - ▶ 에이전트 내 변경은 다른 에이전트에 영향을 미치지 않음
 - ▶ 에이전트 별 데이터 모델을 개별적으로 수정하고 조정할 수 있음
- ▶ 멀티태스킹 지원 가능
 - ▶ 에이전트들을 서로 다른 **Thread, Process, Machine**에 분산 가능
 - ▶ 에이전트에 **IPC** 기능 확장은 자체의 제어 컴포넌트에만 영향

Presentation-Abstraction-Control

Pattern: Liabilities

- ▶ 시스템 설계가 복잡하고 개발이 어려움
 - ▶ 모든 의미적 개념을 에이전트로 구현하면 시스템 구조가 복잡해짐
- ▶ 제어 컴포넌트가 복잡해짐
 - ▶ 제어 컴포넌트들은 통신을 매개함, 다른 에이전트들의 세부 구현에 독립적이어야 함
- ▶ 성능이 떨어짐
 - ▶ 에이전트 간 통신에 부하(**Overhead**)가 발생하면 시스템 효율성에 치명적임
- ▶ 적용가능성(**Applicability**)가 떨어짐, 적용 사례가 적음
 - ▶ 의미적 개념이 작으면, 사용자 인터페이스 유사해지고 적용 가능성이 점점 떨어짐
 - ▶ 의미적 개념이 크고 사람-컴퓨터 상호작용이 필요하다면, 관심의 분리가 명확함



Question?



Seonah Lee
saleese@gmail.com