

# Design Patterns

## Mediator

Dr. Chad Williams  
Central Connecticut State University

# Design pattern: Mediator

- **Category:** Behavioral design pattern
- **Intent:**
  - Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly
- **Motivation**
  - Many reusable objects, but want to avoid direct dependencies to improve reuseability

# Motivation cont.

The diagram shows a form titled "Event RSVP" with four sections. Arrows indicate dependencies between fields across these sections:

- Section 1:** A red asterisk and the word "Required" are pointed to by an arrow from the "Can you attend?" question.
- Section 2:** The question "Can you attend? \*" is followed by two radio button options: "Yes, I'll be there" and "Sorry, can't make it".
- Section 3:** The question "What are the names of people attending?" is followed by a text input field labeled "Your answer".
- Section 4:** The question "Select meal for first person" is followed by three radio button options: "Steak (if selected, display selection for how well done)", "Fish", and "Vegetable lasagna".

Dependency arrows:

- From "Can you attend? \*" to the "Required" label.
- From "Can you attend? \*" to the "Yes, I'll be there" radio button.
- From "Can you attend? \*" to the "What are the names of people attending?" question.
- From "What are the names of people attending?" to the "Your answer" text input field.
- From "What are the names of people attending?" to the "Select meal for first person" question.
- From "What are the names of people attending?" to the "Steak" radio button.
- From "What are the names of people attending?" to the "Vegetable lasagna" radio button.

- Various components are reusable, but run the risk of making individual components very intertwined with dependencies
- Motivation is to extract these dependencies to Mediator to make the components more loosely coupled.

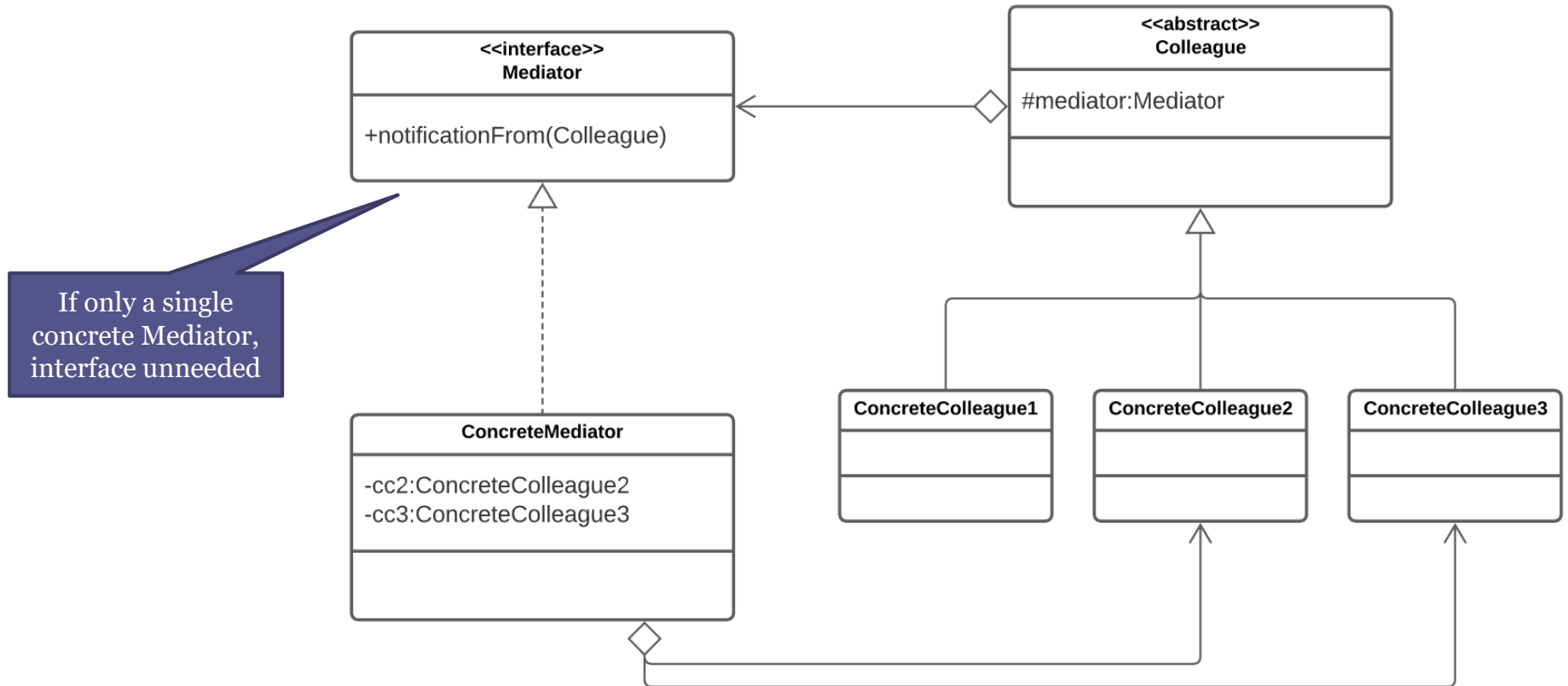
# Applicability

- Use the Mediator pattern when
  - Set of objects communicate in well-defined but complex ways, resulting in interdependencies becoming difficult to understand
  - Reuse is becoming difficult due to references with many other objects
  - Behavior that is distributed between several classes should be customizable without a lot of subclassing

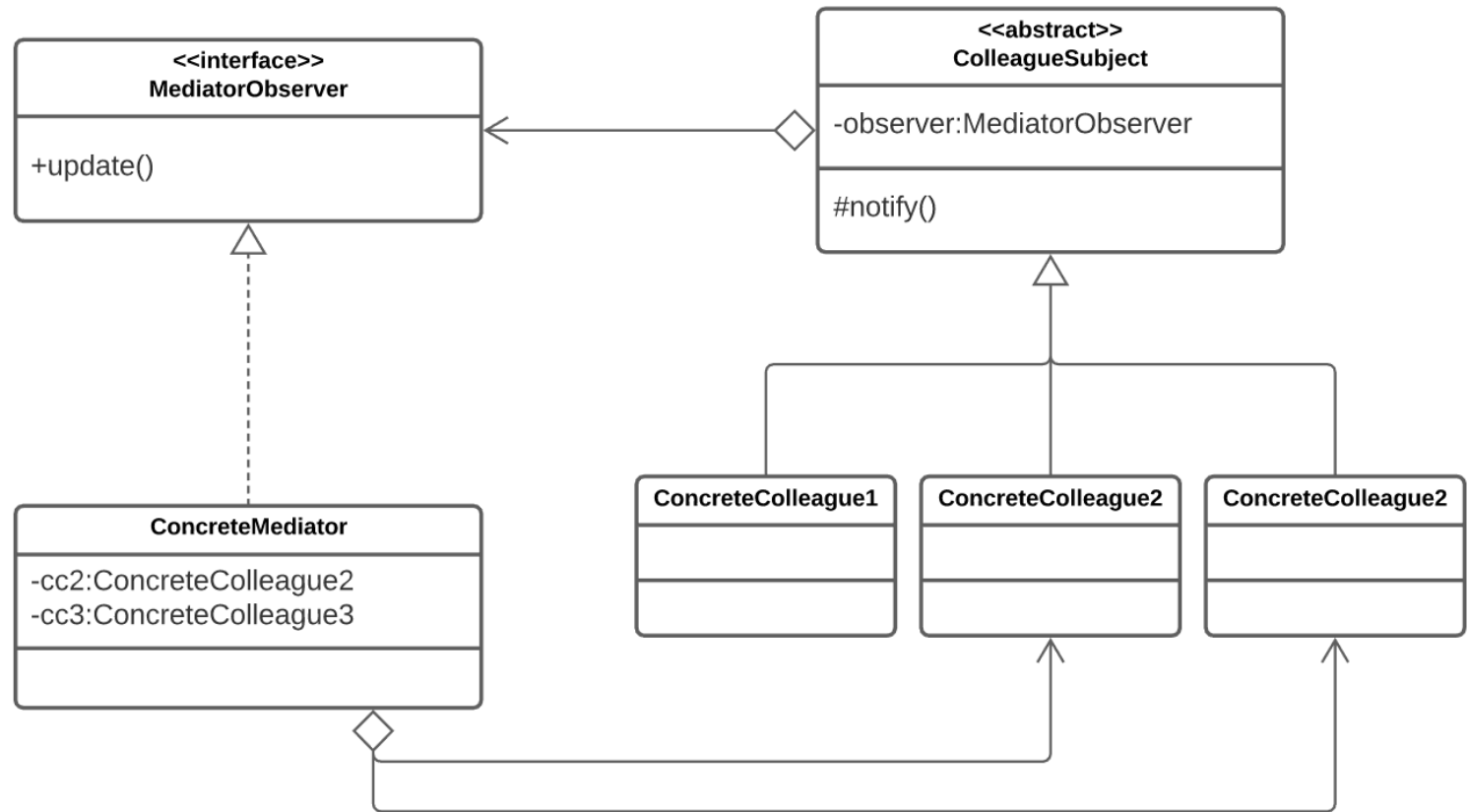
# Participants

- Mediator
  - Defines an interface for communicating with Colleague objects
- Concrete Mediator
  - Implements cooperative behavior by coordinating Colleague objects
  - Knows and maintains its colleagues
- Colleague class
  - Each Colleague class knows its Mediator object
  - Each colleague communicates with its mediator whenever it would have otherwise communicated with another colleague

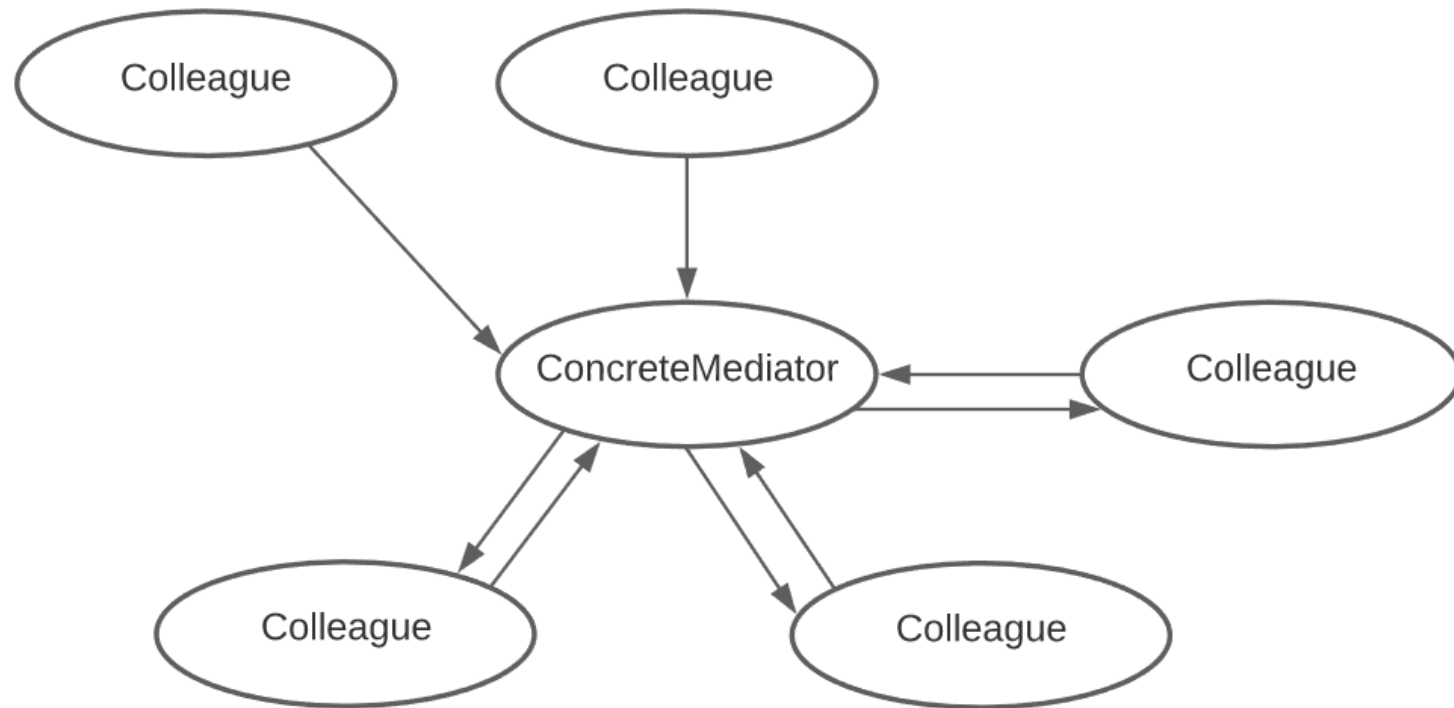
# Mediator structure option 1:



# Mediator structure option 2: Observer pattern



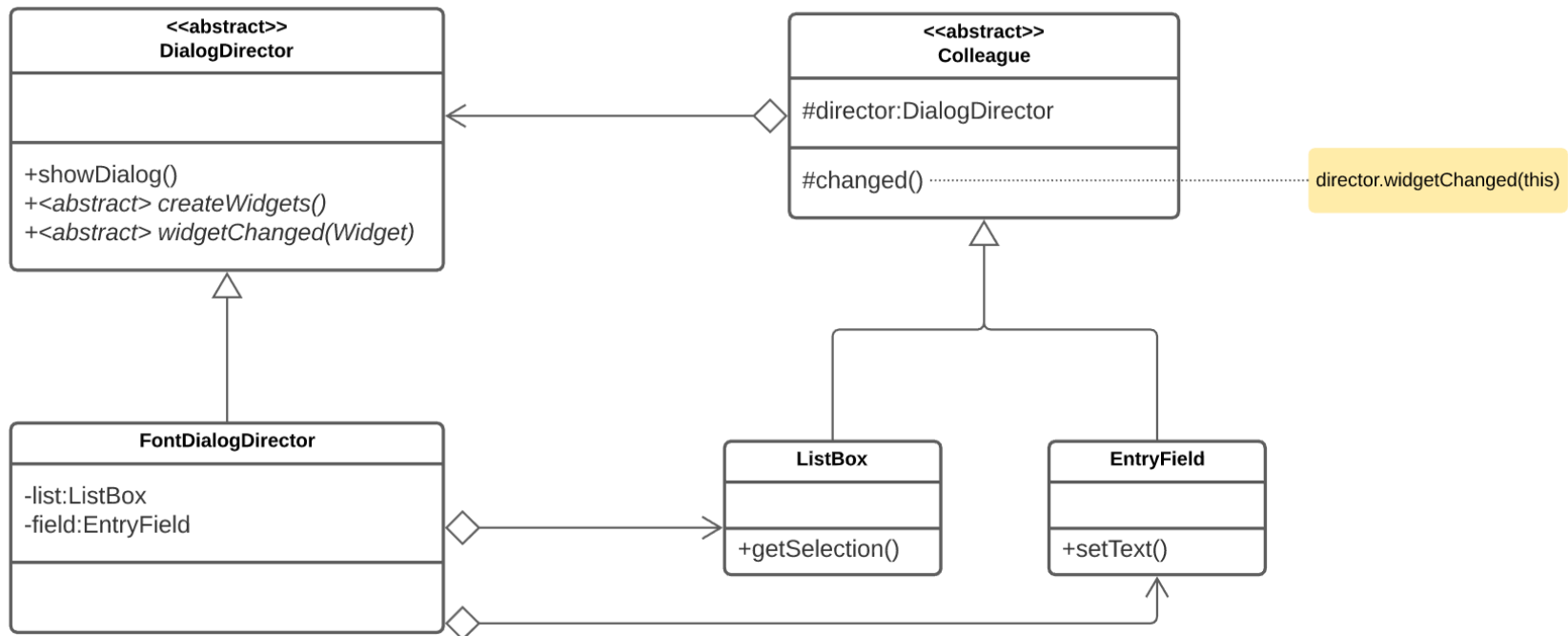
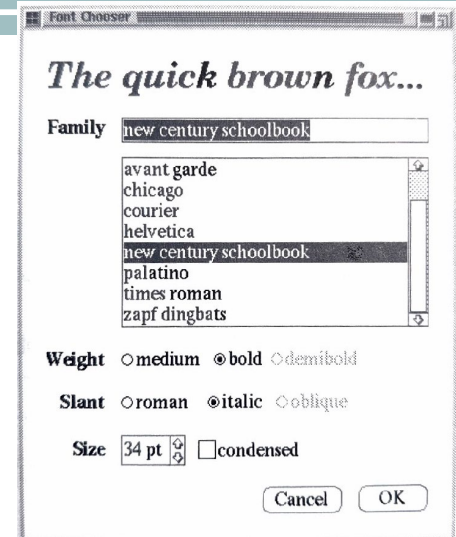
# Typical instance structure



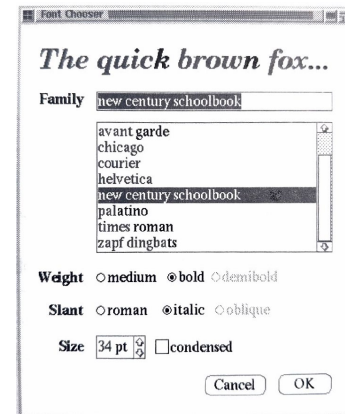
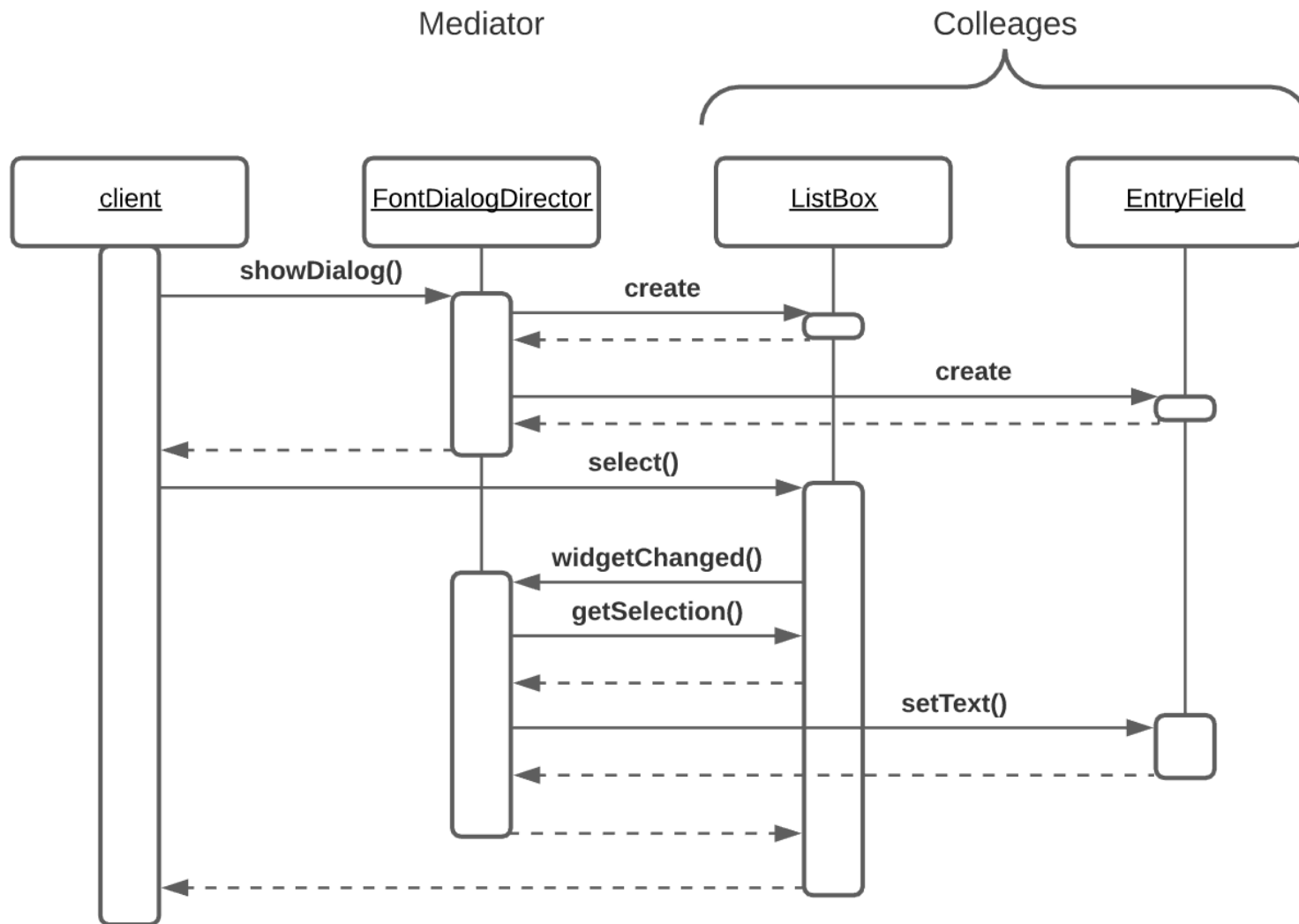
- Collaborations
  - Colleagues send and receive requests from a Mediator object, the Mediator routes requests to the appropriate colleague



# Mediator example



# Interaction pattern



# Consequences

- Mediator pattern has the following benefits and drawbacks
  - **Benefits**
    - Reduces need for Colleague subclassing - Localizes behavior that would have been distributed allowing only Mediator to need subclasses while Colleagues can be used as is
    - Decouples colleagues
    - Simplifies object protocols for many-to-many interactions
    - Abstracts how objects cooperate
  - **Drawback**
    - Centralizes control – simplifies colleagues, but increases complexity of Mediator which as it becomes more complex may be more difficult to maintain

# Related patterns

- Observer
  - Frequently combined having Colleagues communicate with Mediator via Observer pattern
  - Observer is notification mechanism, Mediator pattern is defining specific colleague interaction extracted to Mediator
- Mediator vs Façade
  - Recall Façade's goal was hide interaction with subsystem of components from outside – inside components interacted with each other and did not know about façade
  - Mediator's goal abstract dependencies within subsystem – subsystem components no direct dependencies, all aware of Mediator