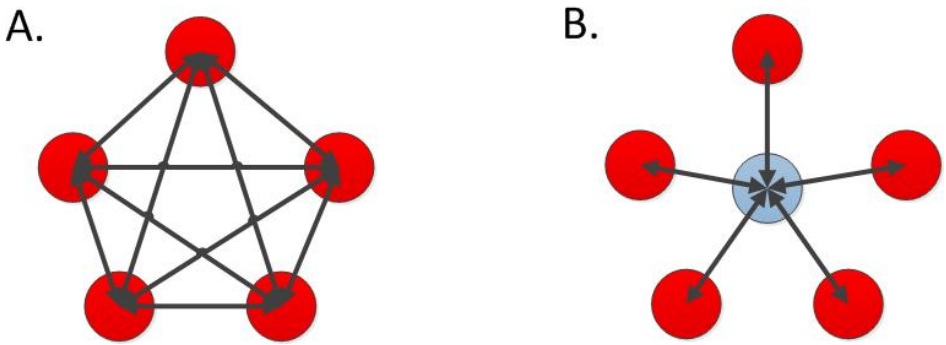
Mediator

A behavioral pattern

Learning goals

1. Learn the idea, structure, and Java implementation of the Mediator design pattern.
2. Learn to apply the Mediator DP in your own programming.

Idea of Mediator



- The Mediator Design Pattern simplifies complex communication channels between interacting objects in a system by providing **a central point of control**.
- In a system composed of many objects, direct communication can lead to dependencies and complexities.
  - Objects directly call methods of other objects. This leads to a tangled web of interactions.
  - Making changes becomes difficult due to tight coupling.

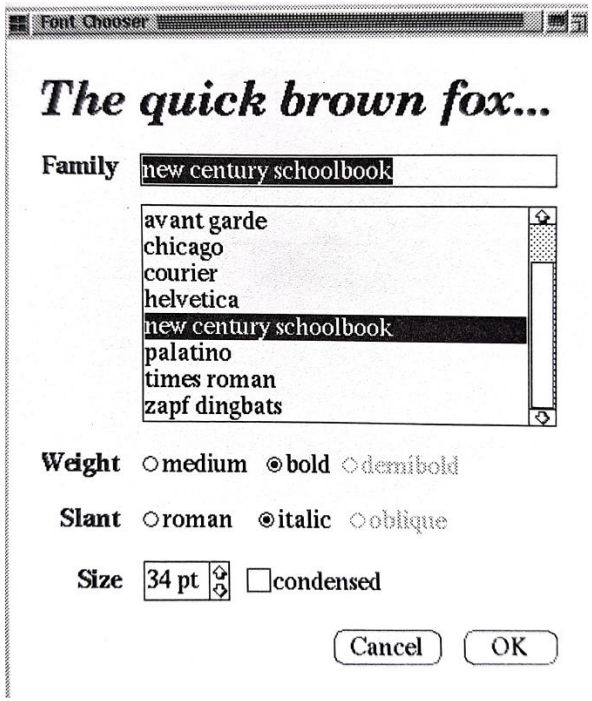
Idea of Mediator

- The Mediator pattern **encapsulates how objects interact**. It prevents objects from referring to each other explicitly.
  - The mediator and the objects communicate via an interface.
  - The objects send their communication needs to the mediator instead of directly to other objects.
- The mediator is responsible for **controlling the workflow** between objects.
- It makes decisions on **when and how to pass messages** or requests between them.
  - The mediator centralizes complex communications and control logic.
  - It forwards the requests to appropriate objects after evaluating the context or state of the system.
- By using the Mediator, the individual **components remain unaware** of the system's evolution or the presence and role of other components.
  - Components are easier to maintain and more flexible to changes.
  - Adding new components or changing interactions becomes less intrusive.

Architecture-level examples

- ATC:
  - Airplane pilots need to make sure that a separation is maintained in the air, and there is no attempt to use the same runway at the same time.
  - Air Traffic Control acts as a Mediator. The pilots only need to communicate with the ATC, not with other planes' pilots.
- Soccer robots:
  - When a robot intends to pass the ball, it may want to inform the teammate (recipient) in advance.
  - The communication with the intended recipient can be done via the mediator.
  - Every robot has an active communication channel to the mediator.
  - The Mediator may design attack patterns and give orders to the individual robots.

Example: dialog window widgets

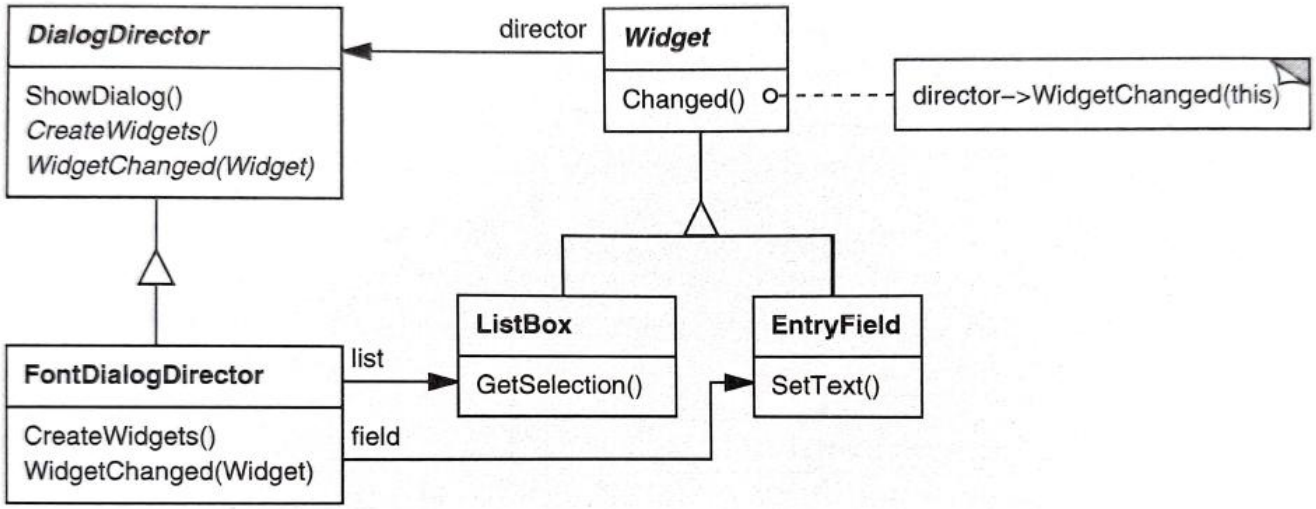


- There are a lot of potential dependencies and communcation needs between objects.

  E.g. a button may get disabled when a field is empty etc.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 273

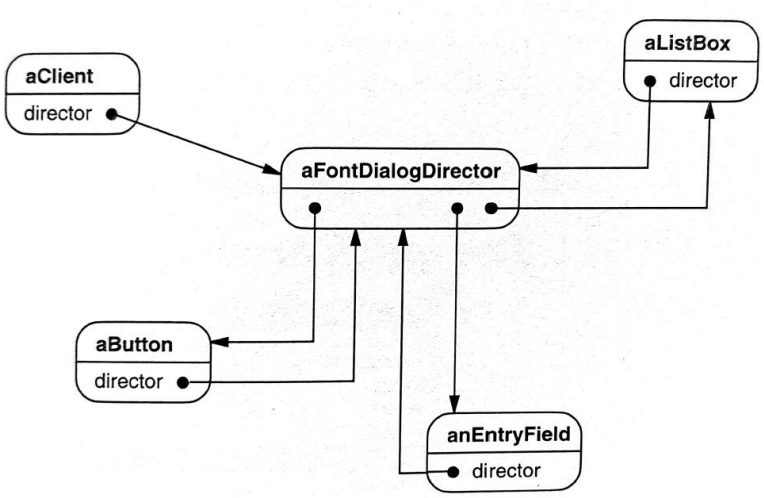
Example: dialog window widgets

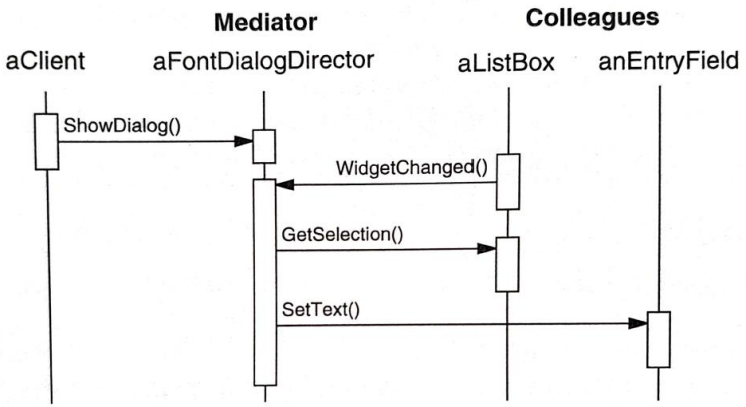


- The Mediator pattern prevents the widgets from knowing each other.
  - They just know the DialogDirector interface.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 275

Example: dialog window widgets

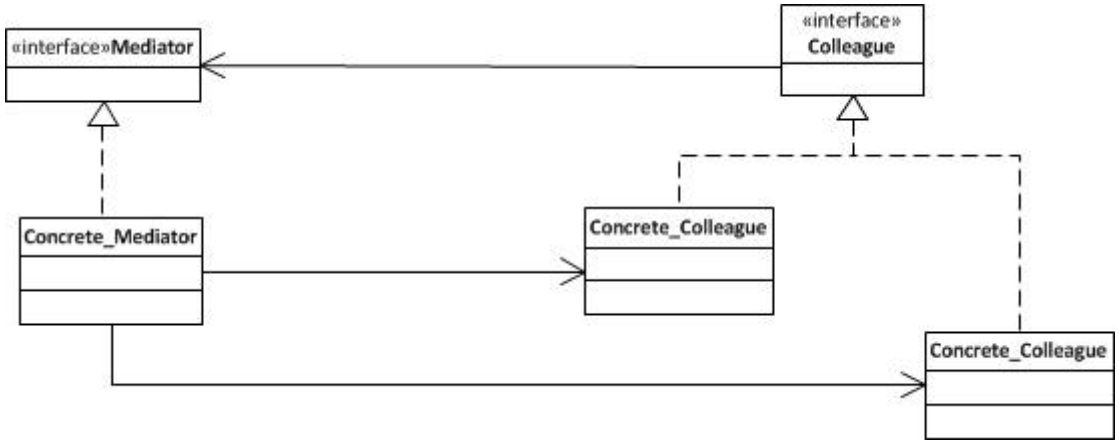




- Communication becomes simple.
  - All widgets know the DialogDirector whose concrete implementation (FontDialogDirector) forwards the requests in a centralized way.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 208

General structure



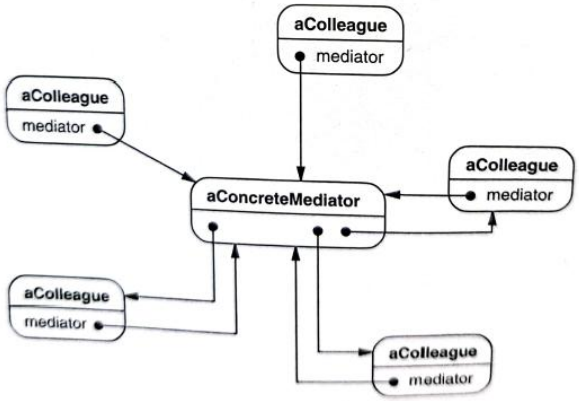


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 276

Roles

- **Mediator**: Acts as the central controlling entity that all participating objects interact with. It knows the protocol for interaction and coordinates the communication between different components.
  - Maintains references to participant objects.
  - Controls the workflow and communication logic.
  - Centralizes complex communications and decisions.
- **Colleague**: Objects that are communicating with each other through the mediator. Each participant knows only about the mediator and not about other colleagues.
- **ConcreteMediator**: An implementation of the Mediator interface. It coordinates the communication between ConcreteColleague objects.
- **ConcreteColleague**: Specific implementations of the Colleague class that engage in the protocol defined by the Mediator.

Practical issues

- The Mediator centralizes communication, which can become a **single point of failure**.
- As the number of colleagues increases, the Mediator might become a **bottleneck**.
  - Specifically, managing concurrent requests through a central mediator can be challenging.
- The logic within the Mediator can become complex as it manages multiple interactions.
  - Refactoring and decomposition of the Mediator into simpler, manageable parts may be necessary.
- Apply the Mediator pattern when the benefits of reduced object dependencies and centralized control outweigh the potential drawbacks.