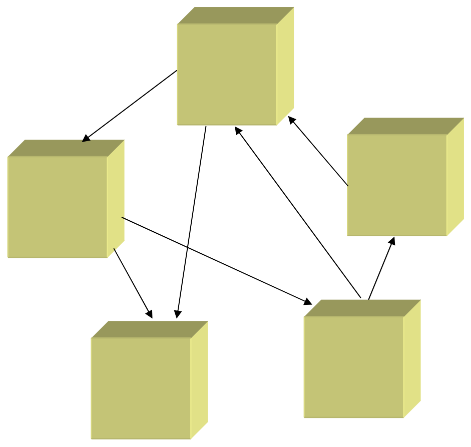
Mediator

GoF Behavioral Pattern
Responsibility pattern

A mediator is responsible for controlling and coordinating the interactions of a group of objects

Removing dependencies with mediators

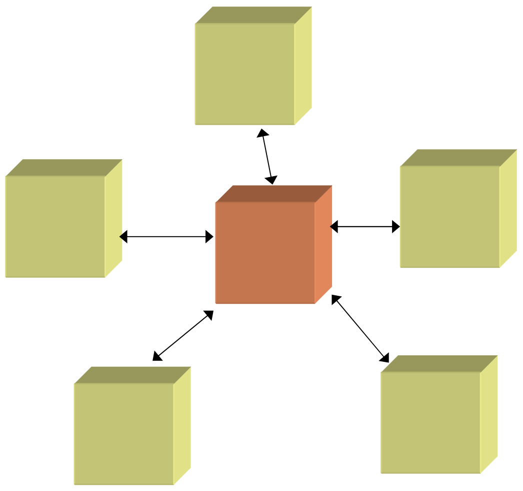
Set of strongly interacting objects



Problems:

- interdependencies are unstructured and difficult to understand
- distributed behavior between several classes cannot be customized or extended without a lot of work, e.g. by subclassing all participating objects
- participants cannot be used in other contexts – no reusability

Mediator



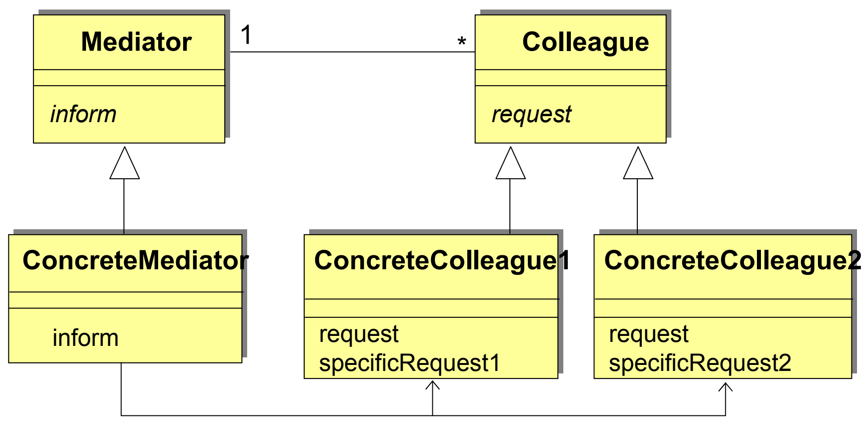
Advantages:
- limits required subclassing to extend: when extending, subclass only the mediator.
- decouples objects, reuse
- simplifies communication (many-to-one instead of many-to-many)
- co-operation abstraction encapsulated to an object, promotes understandability.

Problem: centralized control (mediator may become monolithic)

Mediator intent and problem

- **Intent**
  - Define an object that encapsulates how a set of objects interact.
  - Mediator promotes loose coupling by keeping objects from referring to each other explicitly
  - Lets you vary their interaction independently.
- **Problem**
  - We want to design reusable and maintainable components, but dependencies between the potentially reusable pieces demonstrates the "spaghetti code" phenomenon. You get …
    - “All or nothing” –reuse.
    - “Change one and fix the rest” -maintenance

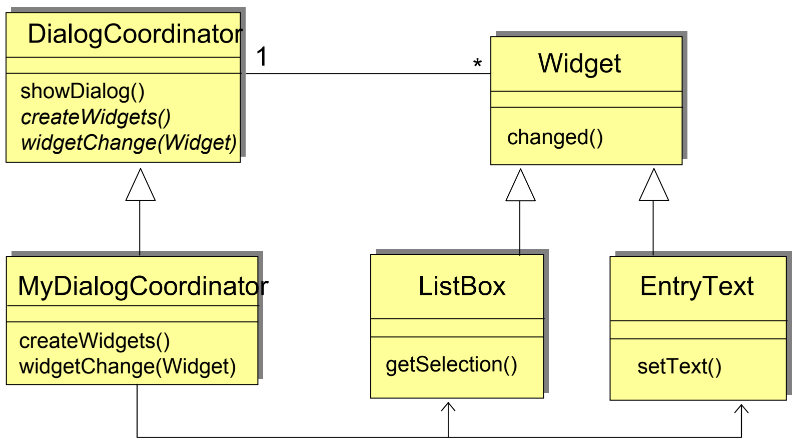
Mediator design pattern



Participants

- Mediator
  - Defines an interface for communicating with Colleague objects
  - Typically mediator is informed of some event or situation
- ConcreteMediator
  - Implements cooperative behavior by coordinating Colleague objects
  - Knows and maintains its colleagues
- Colleague classes
  - Each Colleague class knows its Mediator object
  - Each colleague communicates with its mediator whenever it would have otherwise communicated with another colleague
  - Offers services (requests) to mediator
  - There may be requests that are common to all colleagues, as well as specific ones.

Example – a dialog window



Discussion

- Partitioning a system into many objects generally enhances reusability, but proliferating interconnections between those objects tend to reduce it again.
- The mediator object
  - encapsulates all interconnections
  - acts as the hub of communication
  - is responsible for controlling and coordinating the interactions of its clients
  - promotes loose coupling by keeping objects from referring to each other explicitly.
- The Mediator pattern promotes a "many-to-many relationship network" to "full object status". Modeling the inter-relationships with an object
  - enhances encapsulation
  - allows the behavior of inter-relationships to be modified or extended through subclassing.

Mediator vs. Facade

- Facade (normally) does not add any functionality, Mediator does
- Subsystem components are not aware of Facade
- Mediator's colleagues are aware of Mediator and interact with it

- Both Mediator and Façade are abut imposing policy.
  - Façade is used to impose policy ‘from above’, the policy is clearly visible.
    - Everyone agrees to use the façade instead of the objects beneath it.
    - The use of façade is visible and constraining
  - Mediator imposes policy ‘from below’, the policy is hidden.
    - The policy imposed is a fait accompli rather than a convention
    - The use of Mediator is invisible and enabling