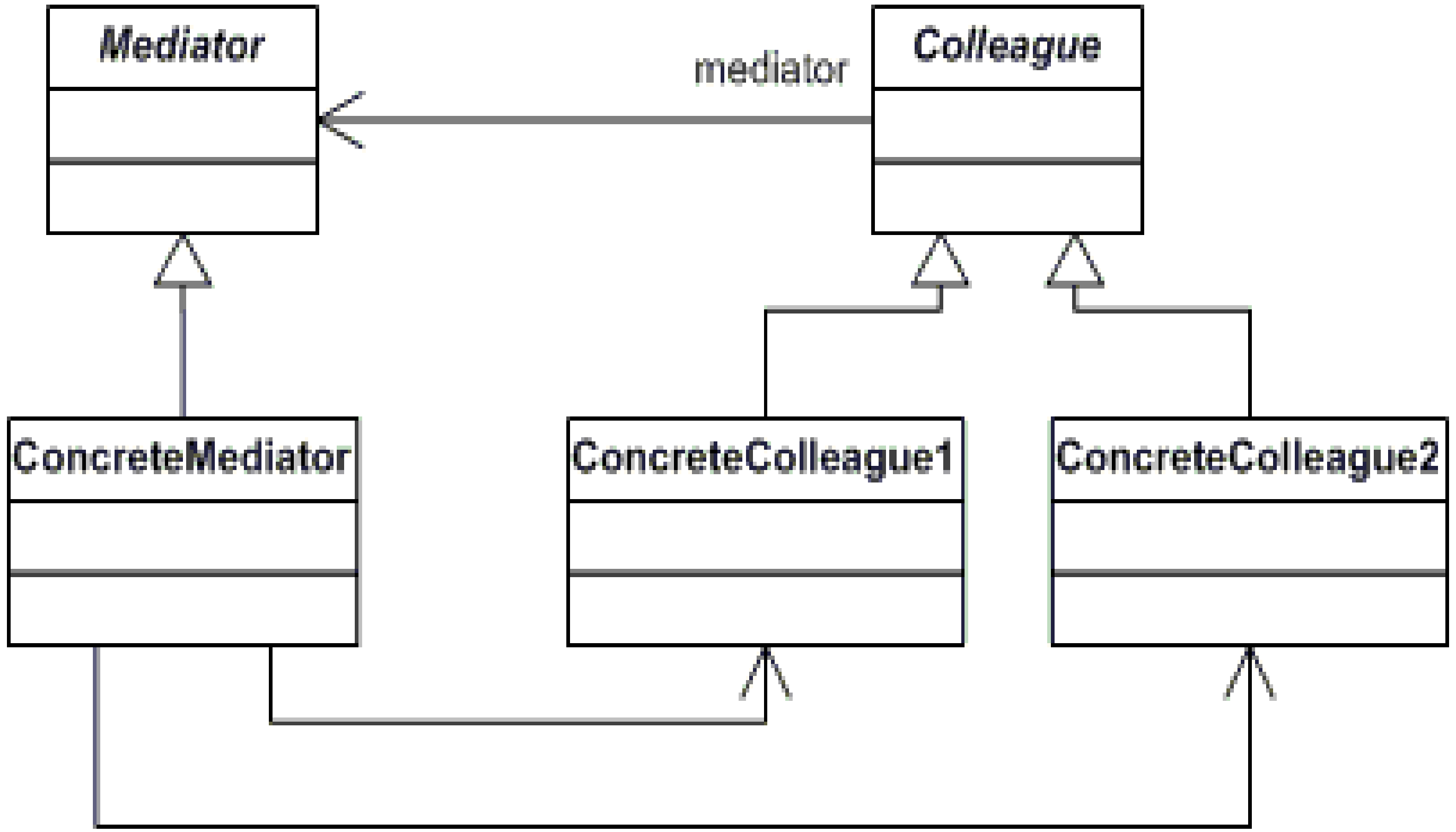
Mediator Pattern

- Define an object that encapsulates how a set of objects interact
- Promotes loose coupling by keeping objects from referring to each other explicitly
- Lets you vary their interaction independently
- Considered behavioral because it defines simplified communication between classes (communication is done via methods which constitute behavior)

UML


Mediator
mediator
Colleague
ConcreteMediator
ConcreteColleague1
ConcreteColleague2

Participants

- Mediator
  - defines an interface for communicating with Colleague objects
- 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

Example: GUI Libraries

- The mediator example is one pattern that is already used in many applications.
- One of the examples is represented by the Dialog classes in GUI applications frameworks.
- A Dialog window is a collection of graphic and non-graphic controls.
- The Dialog class provides the mechanism to facilitate the interaction between controls.
  - For example, when a new value is selected from a ComboBox object a Label has to display a new value.
  - Both the ComboBox and the Label are not aware of each other structure and all the interaction is managed by the Dialog object.
  - Each control is not aware of the existence of other controls.

Example: Chat Application

•The chat application is another example of the mediator pattern.
•In a chat application we can have several participants. It's not a good idea to connect each participant to all the others because the number of connections would be really high, there would be technical problems due to proxies and firewalls, etc... .
•The most appropriate solution is to have a hub where all participants will connect; this hub is just the mediator class.

Chat Example

Participants:

- Chatroom(Mediator) - Defines the interface for interacting with participants
- ChatroomImpl (ConcreteMediator) - implements the operations defined by the Chatroom interface. The operations are managing the interactions between the objects: when one participant sends a message, the message is sent to the other participants.
- Participant(Collegue) - defines an interface for the participants.
- HumanParticipant, Bot (ConcreteCollegue) - implements participants; the participant can be a human or a bot, each one having a distinct implementation but implementing the same interface. Each participant will keep only a reference to the mediator.

Consequences

Advantages

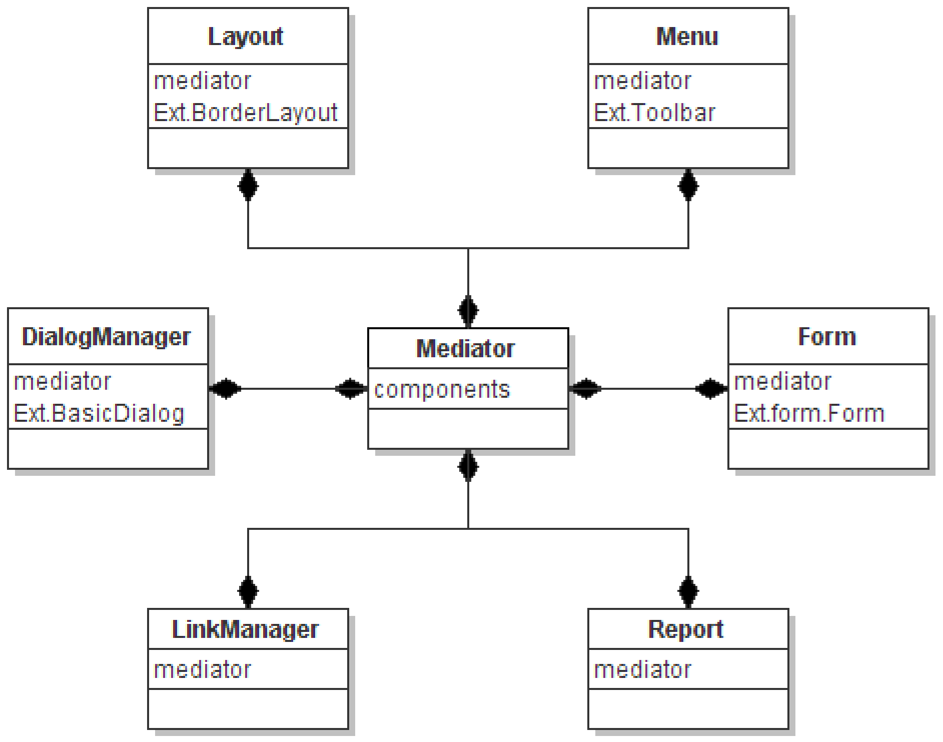
•Comprehension - The mediator encapsulates the logic of mediation between the colleagues. From this reason it's easier to understand this logic since it is kept in only one class.
•Decoupled Colleagues - The colleague classes are totally decoupled. Adding a new colleague class is very easy due to this decoupling level.
•Simplified object protocols - The colleague objects need to communicate only with the mediator objects. Practically the mediator pattern reduces the required communication channels (protocols) from many to many to one to many and many to one.
•Limits Subclassing - Because the entire communication logic is encapsulated by the mediator class, when this logic needs to be extended, only the mediator class needs to be extended.

Consequences

Disadvantages

Complexity - in practice the mediator tends to become more and more complex. A good practice is to take care to make the mediator class responsible only for the communication part. For example, when implementing different screens the the screen class should not contain code which is not a part of the screen operations. It should be put in some other class.

GUI Mediator UML



Mediator

•The mediator pattern is used to take the role of a hub or router and facilitates the communication between many classes.
•A similarity can be made with database systems. The mediator transforms a hard to implement relation of many to many, where each call has to communicate with other classes - into 2 relations: many to one and one to many, where the communication is handled by the mediator class.

Related Patterns

•Facade Pattern - a simplified Mediator becomes a Facade pattern if the Mediator is the only active class and the colleagues are passive classes. A Facade pattern is just an implementation of the Mediator pattern where Mediator is the only object triggering and invoking actions on passive colleague classes. The Facade is being call by external classes.

•Adapter Pattern - the Mediator pattern just "mediates" the requests between the colleague classes. It is not supposed to change the messages it receives and sends; if it alters those messages then it is an Adapter pattern.

•Observer Pattern - the Observer and Mediator are similar patterns, solving the same problem. The main difference between them is the problem they address. The Observer pattern handles the communication between Observers and Subject(s) – it distributes information. With the Mediator pattern, the Mediator class centralizes communication between objects that need to talk.