The Mediator Pattern

Emeka Egbuonye

CSPP 51023
March 02,2010

The Mediator pattern provides a central hub to guide the interactions between many objects

- The Mediator is a behavioral design pattern that provides a central hub to guide the interactions between many objects. According to *Gamma et al,* the intent of the Mediator pattern is to

  "Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently." (Gamma et al., 1995)

- This pattern is considered to be a behavioral pattern due to the way it can alter the program's running behavior

Key participants in the Mediator pattern

Mediator	Defines the interface for communication between Colleague objects
Concrete Mediator	Implements the Mediator interface and coordinates communication between Colleague objects • It is aware of all the Colleagues and their purpose with regards to inter communication.
Concrete Colleague	Communicates with other Colleagues through its Mediator

Similar / Related pattern(s)

- **Facade**: Mediator is similar to Facade because it abstracts functionality of existing classes
- **Observer:** Mediator and Observer are competing patterns. Their differences → Observer distributes communication by introducing “observer” and “subject” objects; Mediator object encapsulates the communication between other objects

Problems associated with large number of classes can be solved by the “Mediator” pattern

Problems that can be solved by the “Mediator”

- **A vicious circle created by too many interactions between objects to the point that every object has a relationship with every other object**
  - An object may be unable to work without the support of a lot of other objects → resulting in same mess you just tried to un-entangle!
  - Where the behavior is shared among different objects, a simple change may necessitate defining a lot of subclasses
- **Communication between a large number of classes**
  - As more classes are developed in a program, (especially during maintenance and/or refactoring), communication between these classes can become more complex
    - ✓ Results in increased difficulty in maintaining and reading the program
- **Nee d for efficient system design**
  - Designing reusable components are more efficient but dependencies between the potentially reusable pieces demonstrate the “spaghetti code” phenomenon

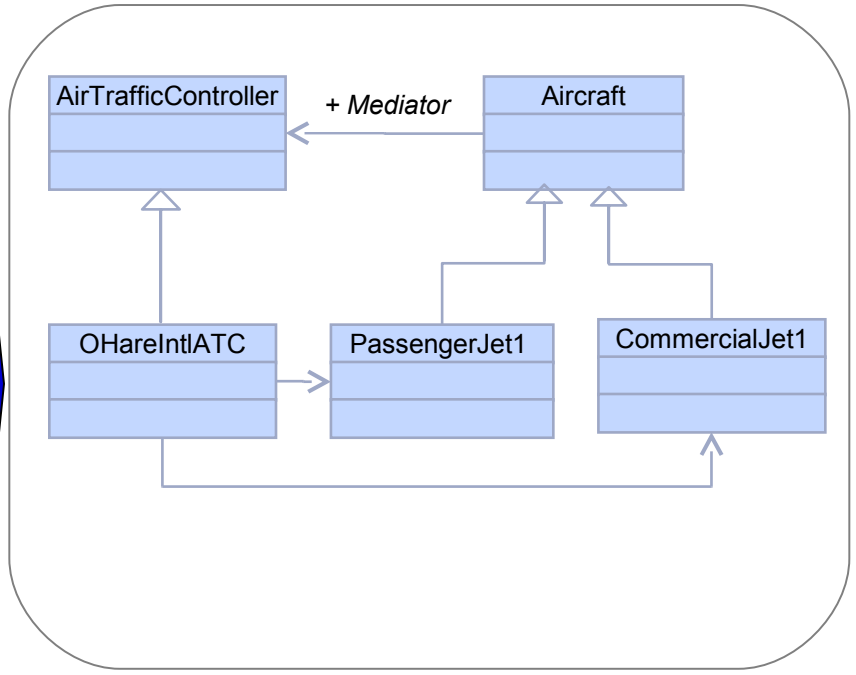
Solutions offered by the “Mediator” pattern

- **Helps to model a class whose object at run-time is responsible for controlling and coordinating the interactions of a group of other objects**
  - Communication between objects is encapsulated with a mediator object
- **Boosts system efficiency as objects no longer need to know about each other, they just need to know their Mediator**
  - As a result, these objects no longer communicate directly with each other, but instead communicate through the mediator
- **Increases design freedom**
  - Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and allowing the designer to vary their interaction independently

A number of real life examples demonstrates the application of the “Mediator” pattern

Example	Detail
Air Traffic Control e.g., O’Hare	➢ No air traffic controllers means every aircraft must know about every other aircraft to avoid collisions • Impossible to all other aircrafts in the vicinity ➢ As a remedy, air traffic controller acts as the Mediator • Aircrafts do not need to communicate with each other directly • All aircrafts send the ATC their flight data • The ATC decides other aircrafts to be informed.
The Stock Exchange (e.g., NYSE)	➢ The Stock Exchange acts as the “Mediator” • Traders do not need to know other traders to buy and sell • The exchange matches traders’ deals and settlements
Speaker of the House (e.g., U.K. Parliament)	➢ The Speaker of the House of Commons acts as the “Mediator” • Presides over the House's debates, determining which members may speak • Maintains order during debates and may punish members who break the rules of the House • MPs (colleagues) may never address each other directly

UML structure of the Air Traffic Control example



Code for stock exchange example (1/2)

The stock exchange acts like a mediator and the traders do not need to know other traders and services provided by them to make a deal. The traders have their own responsibilities as buyers and sellers and it is stock exchange's responsibility to match their deals and get the settlement done

Let's look at the code part:

```
/**
    * StockExchange – this is the mediator class
*/
public class StockExchange {

public static void doTransaction (String typeOfTransaction, int quantity, Scrip scrip, Trader trader) {

Transaction transaction = new Transaction(typeOfTransaction, quantity, scrip, trader);
// try and match the current transaction
// with the ones saved in DB and find out
// whether a counter transaction is there or
// are there many such transactions which could
// fulfill requirement of this transaction.
matchTransaction(transaction)

}
public static getPrice (Scrip scrip) {

// try and match this transaction with all
// the saved ones. If they match till whatever extent
// trade for that. Then save, with status Traded for
// number of shares traded and save the rest as New.

}

}// End of class
```

Code for stock exchange example (2/2)

```
/**
* Trader1 – this trader wants to sell 100 shares of company XYZ
*/

public class Trader1 {

public void doTransaction (String typeOfTransaction, int quantity) {

int expectedPrice = 200;
Scrip scrip = new Scrip("XYZ");
int price = StockExchange.getPrice(scrip);
if(typeOfTransaction.equals("SELL")){

 if(price >= expectedPrice){
StockExchange.doTransaction("SELL", 100, scrip, trader1);
}

}else if(typeOfTransaction.equals("BUY")){

if(price <= expectedPrice){
StockExchange.doTransaction("BUY", 100, scrip, trader1);
}

 }

}

}// End of class
```

```
/**
* Trader2 – this trader wants to buy 100 shares of company XYZ
*/

public class Trader2 {

public void doTransaction (String typeOfTransaction, int quantity) {

int expectedPrice = 200;
Scrip scrip = new Scrip("XYZ");
int price = StockExchange.getPrice(scrip);
if(typeOfTransaction.equals("SELL")){

 if(price >= expectedPrice){
StockExchange.doTransaction("SELL", 100, scrip, trader2);
}

}else if(typeOfTransaction.equals("BUY")){

if(price <= expectedPrice){
StockExchange.doTransaction("BUY", 100, scrip, trader2);
}

}

}

}// End of class
```

Conclusion

The main features of a mediator (based on earlier example code & UML illustration):

- Responsibilities to individual objects
- Mediator is the only smart delegating object
- Transparency for other objects
- If more objects join in, only place of registration is Mediator, other objects do not need to know anything about the new object
- The code becomes very maintainable

It would be difficult to deny the sheer simplicity and usefulness of the Mediator pattern. By providing the loose coupling between Colleague objects and centralizing the interaction logic in the Mediator, changes are more easily facilitated, and code clarity is heightened