

# Mediator Design Pattern

---

PATRON DE DISEÑO MEDIADOR



/luisdavaría

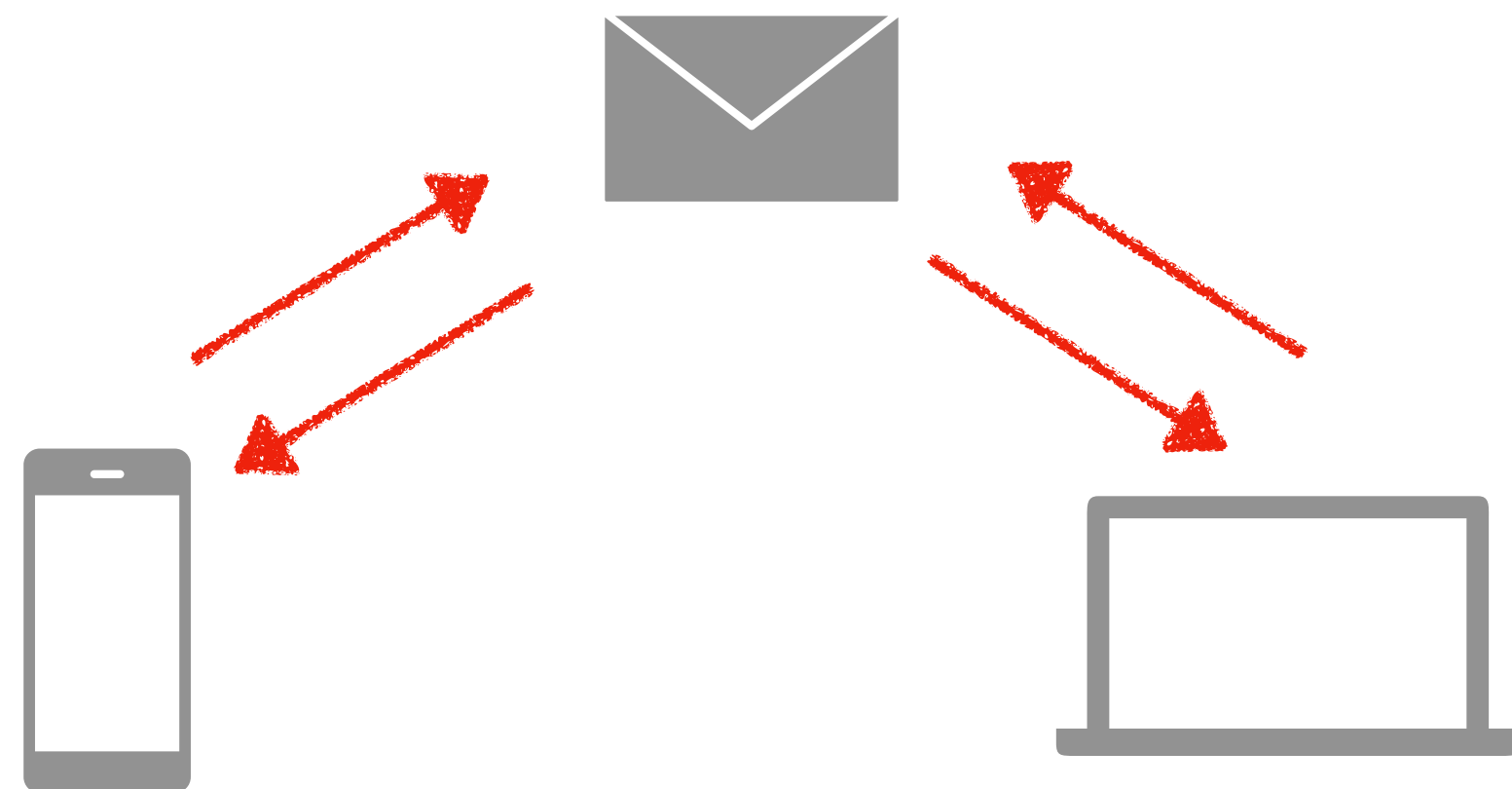


/Davaría



# Mediator

"Facilita la comunicación entre objetos"



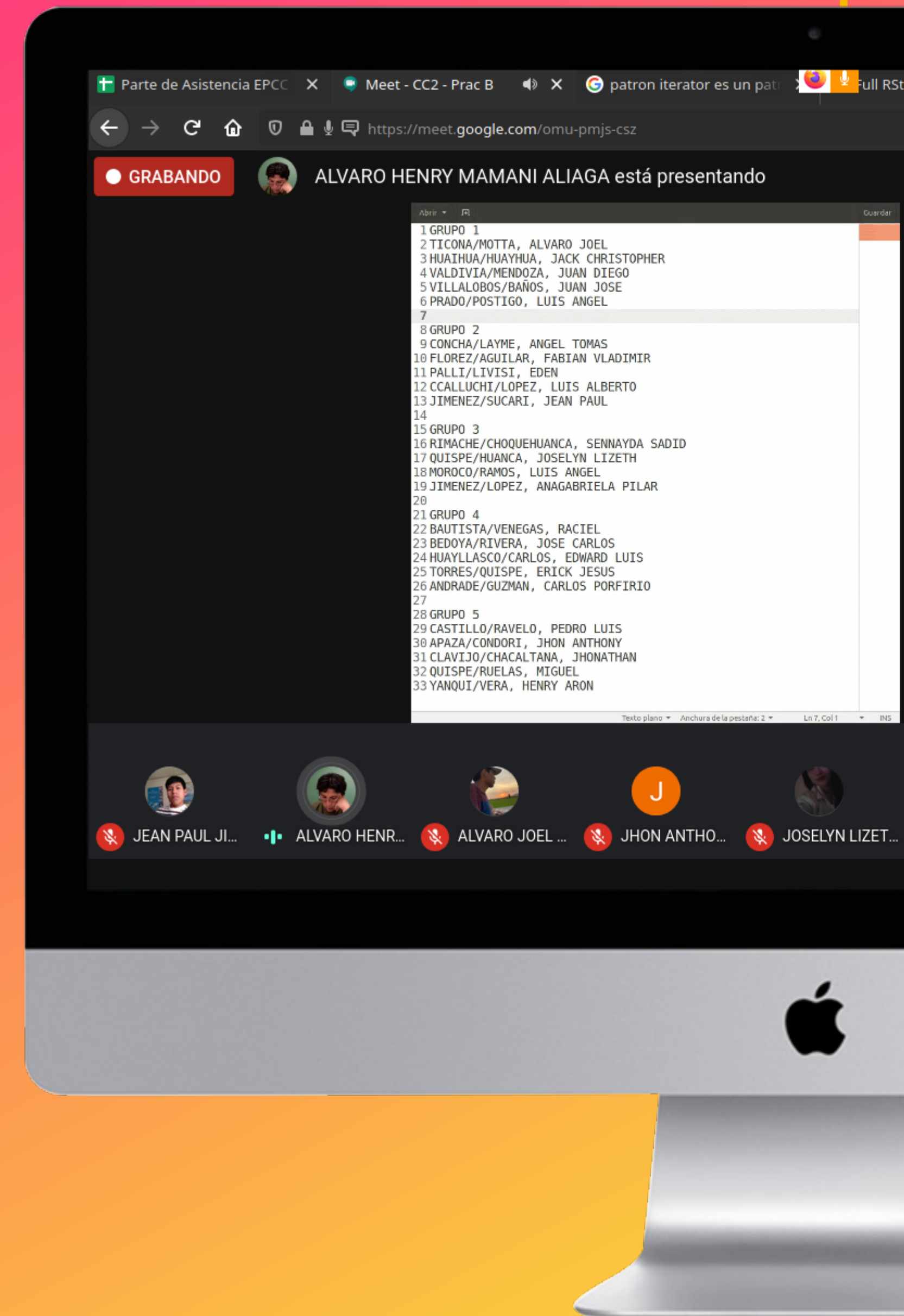
Objeto que encapsula cómo un conjunto de objetos interactúan



Variar interacción de forma independiente



Patron de diseño de tipo Comportamiento





# Programando Google Meet

Un ejemplo clásico y más adecuado del Patrón de diseño de Mediator.



1

Una sala de chat donde sus participantes pueden interactuar.



2

Los participantes pueden referirse entre si pero estas desaparecerán.



3

Participantes deben referir a un componente central que facilite la comunicación y sea el mediador.


# Clase Abstracta ~ ChatRoom

```
● ● ●  
  
class ChatRoom {  
    public:  
        // Transmitir  
        virtual void broadcast(string from, string  
msg)= 0;  
        // Mensaje  
        virtual void message(string from, string to,  
string msg)= 0;  
};
```





# Clase Person ~ Participantes



```
class Person : public ChatRoom{
    public:
        string          name;
        ChatRoom*       room{nullptr};

        void broadcast(string , string ){};
        void message(string , string , string ){};
        Person(string _name) { this->name = _name;}

        void say(string msg) { room->broadcast(name, msg);}
        void pm(string to, string msg) { room->message(name, to, msg); }

        void receive(string from, string msg) {
            string s{from + ": \"" + msg + "\""};
            cout << "[Chat de " << name << "]" << s << "\n";
        }
};
```

# Clase GoogleMeet

## Mediator

- Objeto que encapsula cómo un conjunto de objetos interactúan
- Variar interacción de forma independiente

```
class GoogleMeet : public ChatRoom
{
    public:
        vector<Person*>    people;
        // Transmitir
        void broadcast(string from, string msg) {
            for (auto p : people)
                if (p->name != from)
                    p->receive(from, msg);
        }
        // Unen al chat
        void join(Person *p) {
            string join_msg = p->name + " se unio al chat";
            broadcast("room", join_msg);
            // La clave esta aqui
            p->room = this;
            people.push_back(p);
        }
        // Mensaje
        void message(string from, string to, string msg) {
            auto target = find_if(begin(people), end(people),
                [&](const Person *p) {
                    return p->name == to;
                });

            if (target != end(people))
                (*target)->receive(from, msg);
        }
};
```



```
int main() {  
    GoogleMeet room;  
  
    Person john("John");  
    Person jane("Jane");  
  
    room.join(&john);  
    room.join(&jane);  
    john.say("Jane hiciste la tarea?");  
    jane.say("oh, hey john");  
  
    Person simon("Simon");  
    room.join(&simon);  
    simon.say("Hola a todos!");  
  
    jane.pm("Simon",  
"Me alegro de que nos hayas encontrado, Simon!");  
  
    return 0;  
}
```



# Google Meet



```
[Chat de John]room: "Jane se unio al chat"  
[Chat de Jane]John: "Jane hiciste la tarea?"  
[Chat de John]Jane: "oh, hey john"  
[Chat de John]room: "Simon se unio al chat"  
[Chat de Jane]room: "Simon se unio al chat"  
[Chat de John]Simon: "Hola a todos!"  
[Chat de Jane]Simon: "Hola a todos!"  
[Chat de Simon]Jane: "Me alegro de que nos ha  
yas encontrado, Simon!"
```

# Ventajas de Mediator

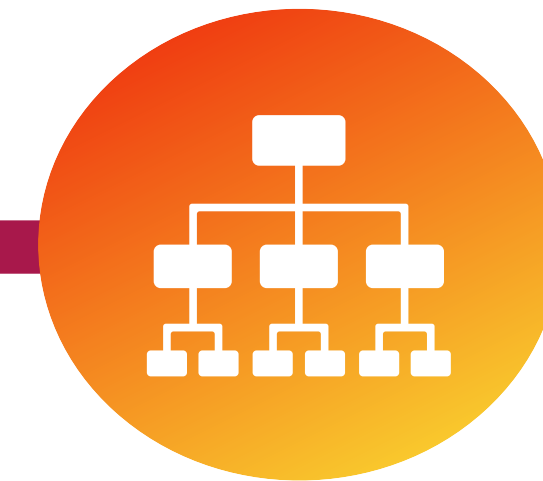


Supera

- Observer funciona en un relación uno a mucho.
- Mediador muchos a muchos.



Reduce la complejidad de la comunicación entre los diferentes componentes de un sistema.



Puede reemplazar cualquier componente en el sistema sin afectar a otros componentes y sistema.



Fondo de color



Tipografías

**TITULOS /Thonburi**

Texto/Menlo