Forwarder-Receiver
Client-Dispatcher-Server

Andrej Thomas Dobrev

Problem

We want to monitor the temperature at our greenhouse from our phone

We have a smart thermostat in the greenhouse that is connected to the internet

We have a simple app on our phone that shows us the temperature at the greenhouse

Code representation

```
class Thermostat
{
    private double temperature { get; private set; }

    public void UpdateTemp(double temp)
    {
              temperature = temp;

              // send temperature data to connected phone
over the
       // internet
    }
}
```

```
class GreenhouseTemperatureApp
{

    private Display display;

    // Gets called when app receives temperature
    // update from the thermostat
    public void ReceiveTempUpdate(double temp)
    {
       Display.SetTemperature(temp);
       Display.update();
    }
}
```

Naive solution - thermostat side

```
class Thermostat
{
    private readonly HttpClient httpClient;

    public Thermostat()
    {
        httpClient = new HttpClient();
    }

    public async Task UpdateTemp(double temp)
    {
        temperature = temp;

        try
        {
            var response = await httpClient.PostAsync(
                "http://<Smartphone_IP_Address>:5000/temperature",
                new StringContent(temp.ToString())
            );

            response.EnsureSuccessStatusCode();
        }
        catch (HttpRequestException e)
        {
            Console.WriteLine($"Error updating temperature: {e.Message}");
        }
    }
}
```

Naive solution - app side

```
class GreenhouseTemperatureApp
{
    private readonly HttpListener httpListener;

    public GreenhouseTemperatureApp()
    {
        httpListener = new HttpListener();
        httpListener.Prefixes.Add("http://*:5000/temperature/");
    }

    public async Task StartListening()
    {
        httpListener.Start();

        while (httpListener.IsListening)
        {
            var context = await httpListener.GetContextAsync();
            var request = context.Request;
            var response = context.Response;

            var temperature = await new System.IO.StreamReader(request.InputStream).ReadToEndAsync();
            RecieveTempUpdate(double.Parse(temperature));

            response.Close();
        }
    }

    public void ReceiveTempUpdate(double temp)
    {
       Display.SetTemperature(temp);
       Display.Update();
    }

}
```

Problems with the naive solution

- thermostat and the app are tightly coupled
- breaks the single responsibility principle
- any changes to the connection will require changing the source code of thermostat/app

```
class Thermostat
{
    private readonly HttpClient httpClient;

    public Thermostat()
    {
        httpClient = new HttpClient();
    }

    public async Task UpdateTemp(double temp)
    {
        temperature = temp;

        try
        {
            var response = await httpClient.PostAsync(
                "http://<Smartphone_IP_Address>:5000/temperature",
                new StringContent(temp.ToString())
            );

            response.EnsureSuccessStatusCode();
        }
        catch (HttpRequestException e)
        {
            Console.WriteLine($"Error updating temperature: {e.Message}");
        }
    }
}
```

Infrastructure

Solution using Forwarder-Receiver

```
class Thermostat
{
    private double temperature { get; private set; }
    private readonly MessageForwarder forwarder;

    public Thermostat(MessageForwarder forwarder)
    {
        this.forwarder = forwarder;
    }

    public async Task UpdateTemp(double temp)
    {
         temperature = temp;
         await forwarder.ForwardMsg(temp.ToString());
    }
}
```

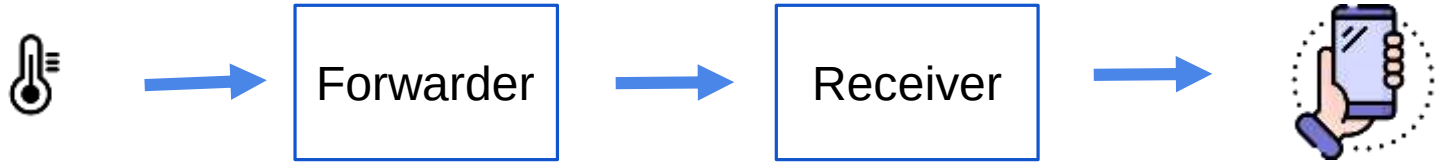
```
class GreenhouseTemperatureApp
{
    private readonly MessageReceiver receiver;

    public GreenhouseTemperatureApp(MessageReceiver receiver)
    {
         this.receiver = receiver;
    }

    public void UpdateDisplayTemp(double temp)
    {
        Display.SetTemperature(temp);
        Display.Update();
    }


    public async Task ListenForTemperatureUpdates()
    {
        while(true)
        {
            double temperature = await receiver.ReceiveMsg();
            UpdateDisplayTemp(temperature);
        {
    }
}
```

Solution using Forwarder-Receiver

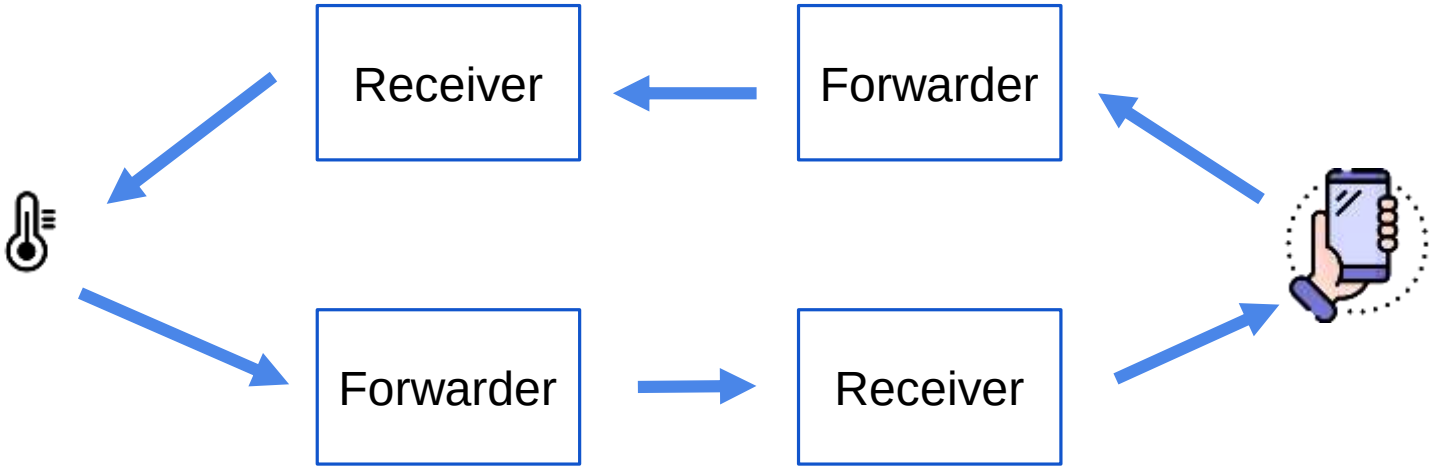
```
class MessageForwarder
{
    private readonly HttpClient httpClient;
    private readonly string serverUrl;

    public MessageForwarder(string serverUrl)
    {
        this.serverUrl = serverUrl;
        httpClient = new HttpClient();
    }

    public async Task ForwardMessage(string message)
    {
        await httpClient.PostAsync(
            serverUrl,
            new StringContent(message)
        );
    }
}
```

```
class MessageReceiver
{
    private readonly HttpClient httpClient;
    private readonly string serverUrl;

    public MessageReceiver(string serverUrl)
    {
        this.serverUrl = serverUrl;
        httpClient = new HttpClient();
    }

    public async Task<string> ReceiveMessage()
    {
        var response = await httpClient.GetAsync(serverUrl);
        return await response.Content.ReadAsStringAsync();
    }
}
```

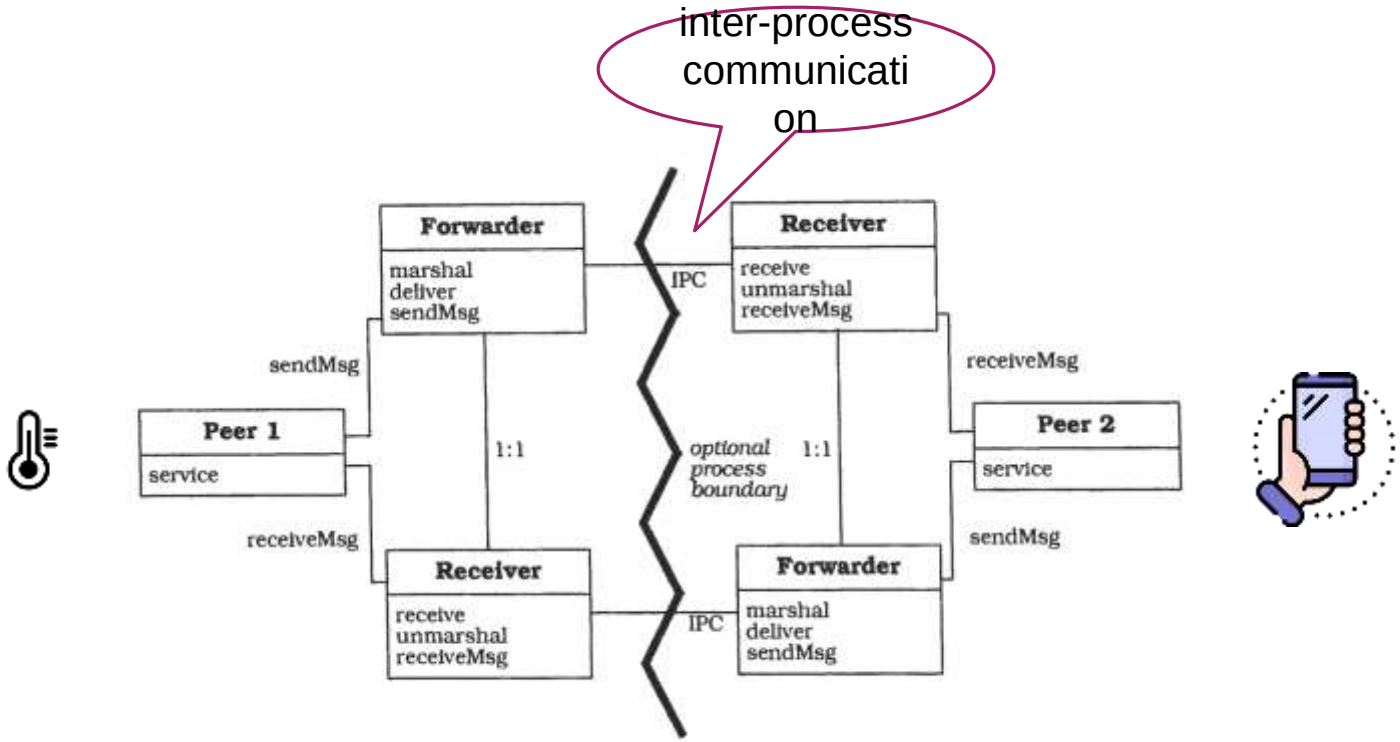
Thermostat sends updates to phone on change



Phone sends requests to thermostat, thermostat responds to the requests.



Forwarder-Receiver Diagram



Forwarder-Receiver summary

Pros:

- Abstraction above concrete form of communication
- Decoupling of the sender (forwarder) and receiver
- Scalability, as multiple receivers can subscribe to messages of one forwarder, allowing for parallel processing
- asynchronous communication

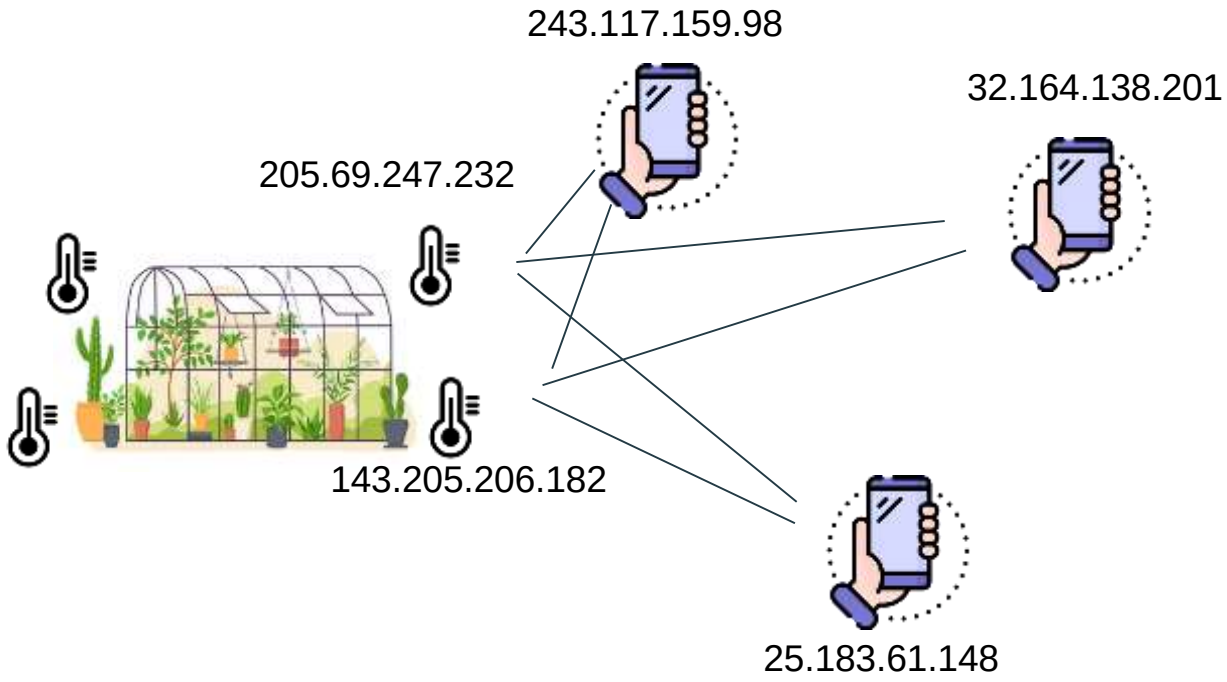
Cons:

- complexity
- possible latency
- synchronization

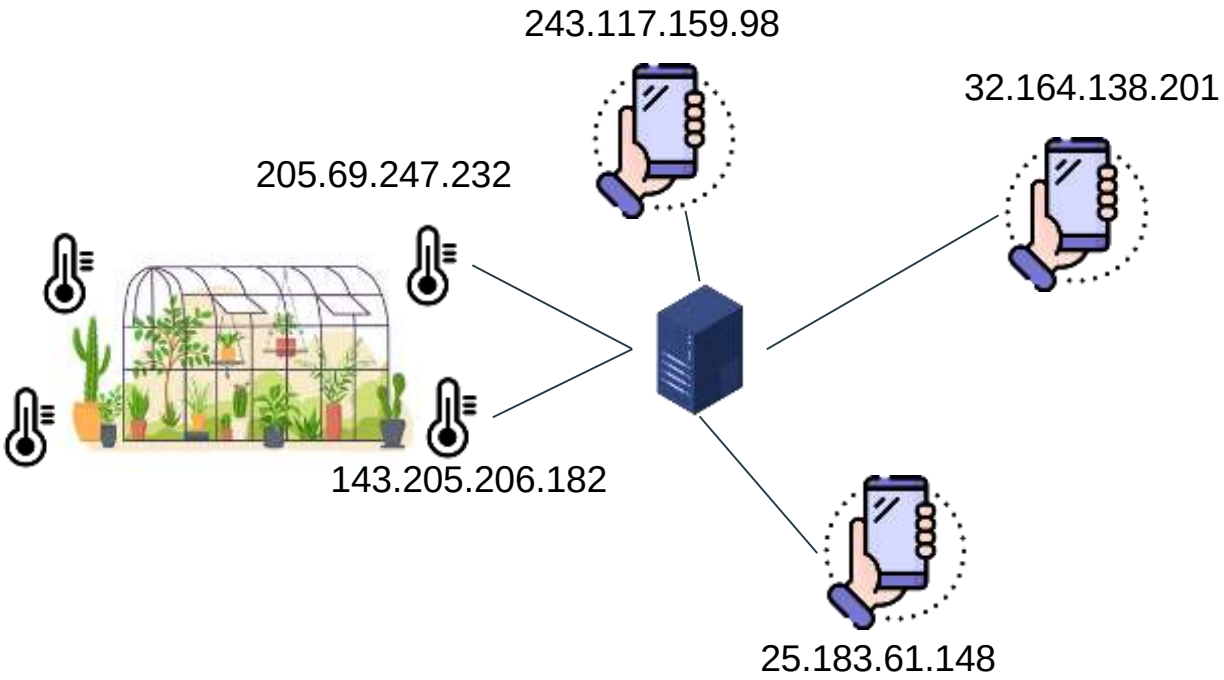
When to use:

- Asynchronous communication - particularly in event-driven architectures and real-time systems
- Distributed systems
- parallel processing
- event handling

Problem continuation



Solution using a dispatcher



Client-Dispatcher-Server responsibilities

Dispatcher

- Routing
- Load Balancing
- Fault Tolerance
- Monitoring and Management
- Security

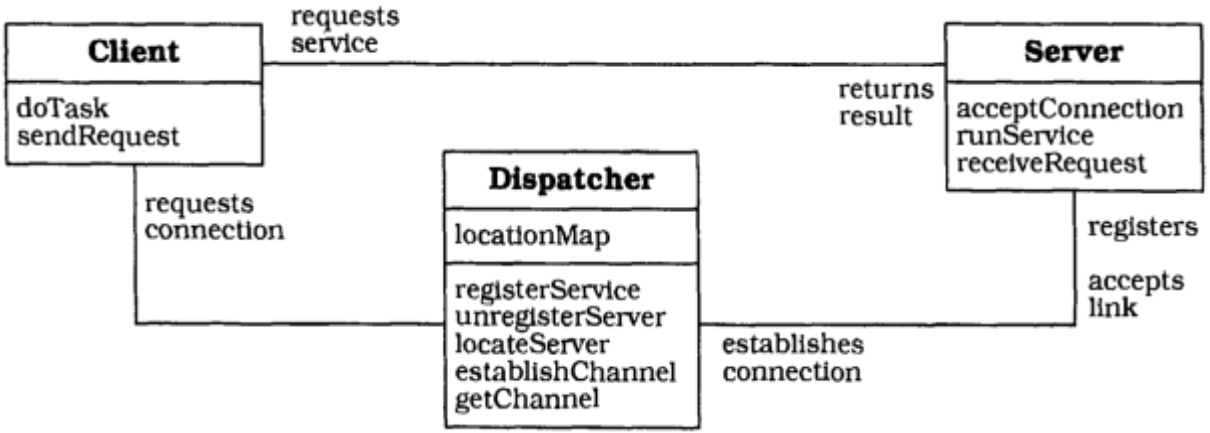
Server

- Provider
- Processor
- Responder (to Client)

Client

- Initiator
- Consumer
- Responder

Client-Dispatcher-Server Diagram



Client-Dispatcher-Server

It is often used with Forwarder-Receiver