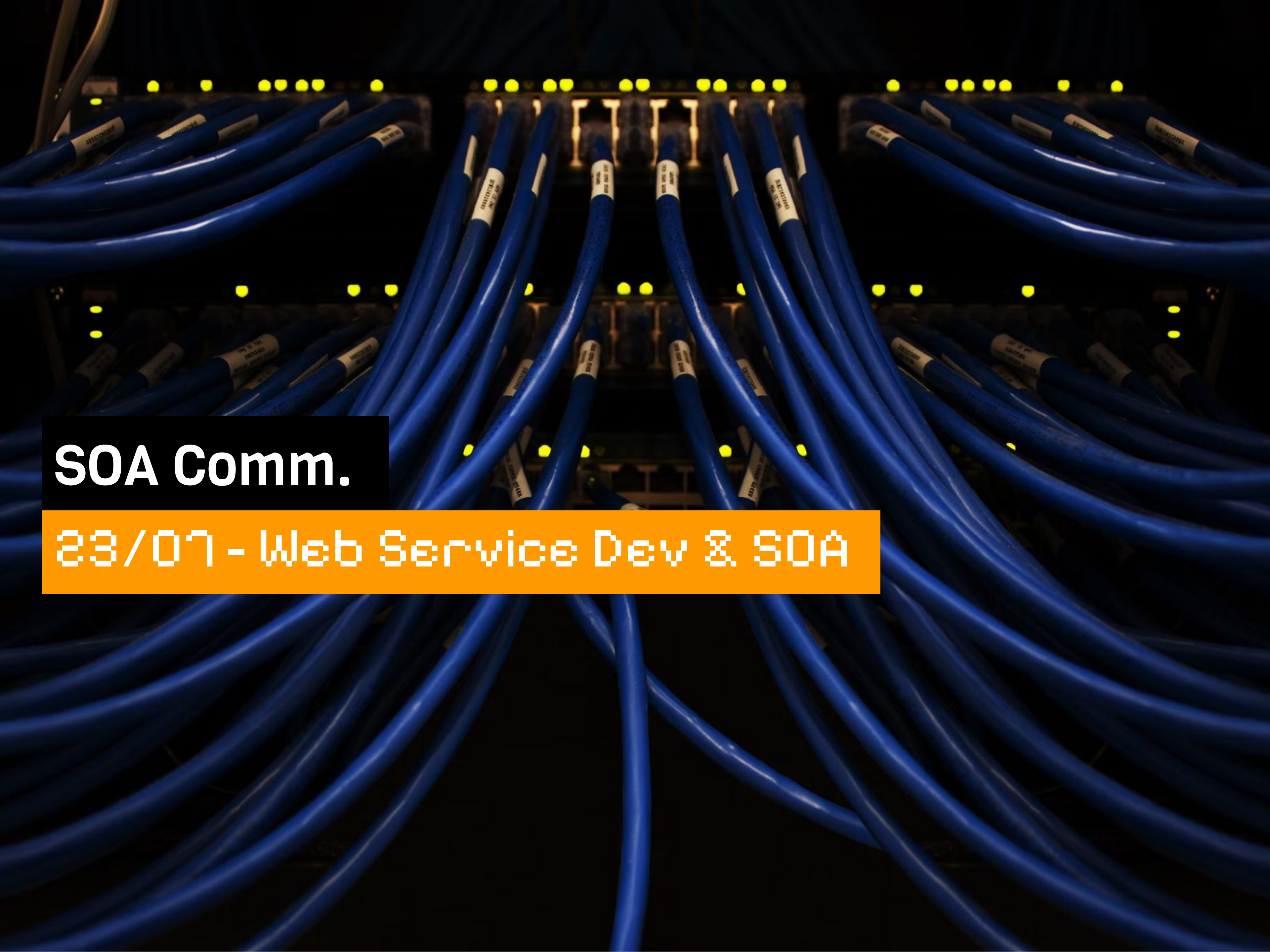


SOA Comm.

23/07 - Web Service Dev & SOA

ADENDA

WHAT'S ON THE MENU? - WEEK 4

I: Voting Details, Model Ans. Tips & Recap

II: SOAP

III: Enterprise Service Bus (ESB)

I: Voting Details, Model Ans. & Recap.

23/07 - Web Service Dev & SOA

Voting Details

<u>**Next week**</u>**, vote** (by attaching the team's choices in the exercise submission) for the following**:**

- Presentation Organisation (within 20 mins):
    - **Self-organised:** Allocate time for Presentation & Live Demo by your own. Declare them in the presentation slide.
    - **Course-organised**: 60% Presentation + 40% Live Demo.

- Presentation Order:
    - **Opt for Reservation:** Those who submit the presentation slide the earliest (anytime before the Week 7 class) = the first to present.
    - **Opt for Random:** Randomised sequences will be announced on Week 6. Will be randomised within the pool of those who are opted for random.

Voting Details

- **Eligibility:** Any team that submit the group exercise.
    - **Submitted this exercise week** = You are in (by default).
    - **Not submitted this exercise week** = Will receive two reminders:
        - 1st Email: After the class.
        - 2nd Email: On this Friday.
        - **No response** to both = **Ineligible.**

- **Estimated Presentation Time: 20 minutes**
    - Slides + Live Demo.
    - **No Q&A**. Missed Something Important in the Slide = Lose Mark (in *C1 & C2*).
    - Overtime/Late Presentation = Lose Mark (in *C3*).
    - **Plan for Failure in Live Demo:** Rationalise/Elaborate How <u>You Achieve All Three Requirements</u> in the Slide = (At least) Get Marks in C1.
    - The slide must have all components as demonstrated in ***the model answer*** + Answer **all the problem statement reqs**.

Model Answer Tips

Here's How You Can Excel at The Exams (& Presentations):

Master the diagrams

Only two will be used: Component Diagram & Communication Diagram

Writing for everything else

(Will have more detail today.)

Practice, Practice & Practice

Use the course **group's exercise** & the **problem statements.**

Aware of time limitation

Time is limited, plan & strategise ahead when answering.

Model Answer Tips

Additional Notes on Exams:

Problem Statement: Similar to the group project problem statements, but **one** requirement. Use all problem statements to practice. Then, try to design apps in App Store/Play Store.

Apps domain: Entertainment, Multimedia **OR** Utilities.

Each exam question Similar to/combined from the group exercise**:**
Exam Question 1: Week 3
Exam Question 2: Week 4
Exam Question 3: Week 5 & 6

Designing Model Answer

Design the System in Which ***Architecturally*** Address the Following:

Self-healing : Design for failure.

If one failure, the other should not be disrupted while one is recovering.

Scalability: Increase reusability.

If one need helps, its clones should be able to join in & help /w the least disruption. (More Next Wk.)

Security: Minimising exposure

If one hacked, an attacker has the least data & control over the system.

Writing Model Answer

Most of you get at least C+ (i.e. No more generic answer). But there is a room for improvement for the presentation and exams:

Knowledge Connection

For C1: Try to connect to the class's topics altogether.

SOA Limitations

For C1: Included in last & this week. Connect them to the rest.

Examples over Name droppings

For C2: Example scenarios are more convincing.

Avoid Misinfo.

For C2: Cross-check the answer before submit. **Lose marks** if found out. (Found last week)

Writing Model Answer

Example (From Last Week)

We chose JSON for its compact data format over XML **to improve communication between Front-End and other components, especially on wireless tablets, which is prone to connectivity issues.** **This applies to both sync. and async. requests.** **Although connectivity problems may still occur with JSON, we'll add a checksum to each request to identify the completeness of the request.**

- **C1: Evidence of knowledge and understanding**: Purple Part = Topics mentioned in Week 2 (applicable to any part of the course). **SOA Limitations:** Pink Part = the analysis from *the limitations of the course* & *outside of the course* (i.e. any protocols, including JSON can suffer from connectivity issues).

- **C2: Persuasive:** Green part uses an example scenario that Front-End are wireless tablets. **Sound:** Wireless devices can prone to connectivity issues (5G, WiFi or otherwise).

Recap: Component Diagram

From last week, some reminders.

Component Diagram

Missing Interface Name

Easy Fix. Make Sure to All (Important) Components Have Them.

Mediator vs Peer-to-Peer

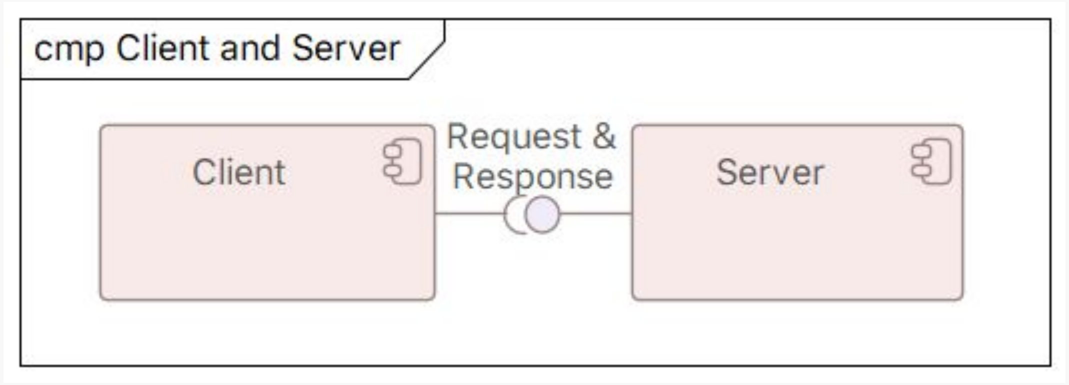
COMPONENT DIAGRAM

Establishing Ground Truth:

- **Interface** = One of the most overused words in SE. For Component Diagram: The place at which independent and often unrelated systems meet and act on or communicate with each other (from Merriam-Webster dict.)

- Component Diagram = Structure Diagram (not behavioural diagram). Focus on the "act on" of the above definition.

Ex (Client-Server):

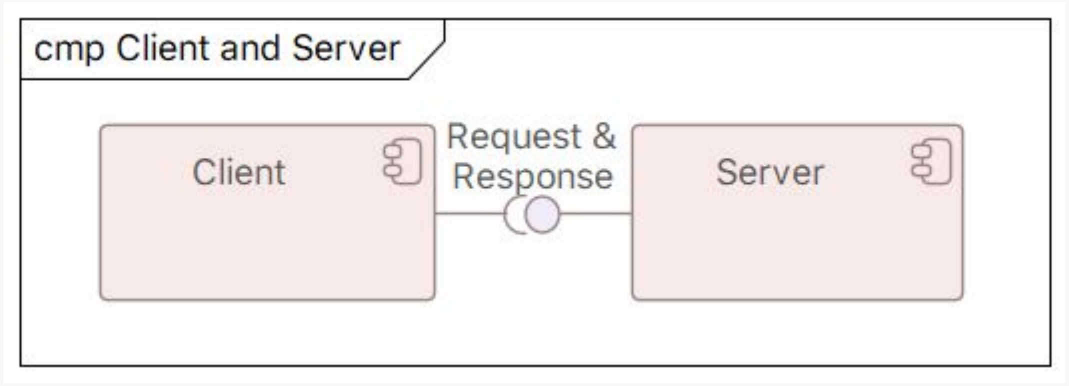


Component Diagram

How to Figure to Determine Provided Interface & Required Interface ?

- **Required Interface =** The one who "**act**" on (i.e. the one that **require** the response).

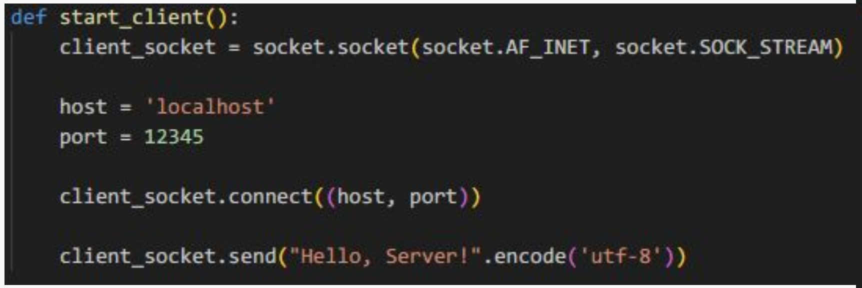
- **Provided Interface =** The one who "**react**" thereafter (i.e. the one that **provide** the response).



Component Diagram

How to Figure to Determine Provided Interface & Required Interface ?

Ex (Code Level : Client - Server in Python):

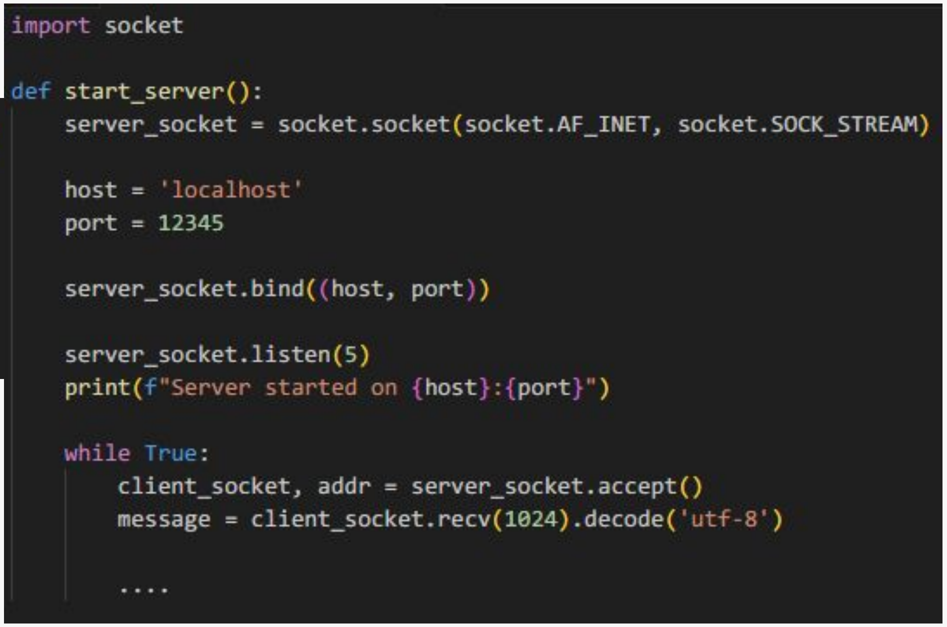


```python
def start_client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    host = 'localhost'
    port = 12345

    client_socket.connect((host, port))

    client_socket.send("Hello, Server!".encode('utf-8'))
```



```python
import socket

def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    host = 'localhost'
    port = 12345

    server_socket.bind((host, port))

    server_socket.listen(5)
    print(f"Server started on {host}:{port}")

    while True:
        client_socket, addr = server_socket.accept()
        message = client_socket.recv(1024).decode('utf-8')

        ....
```

Client: **Act on** a server via sending a request.

Server: Waits for a request from client to **react**.

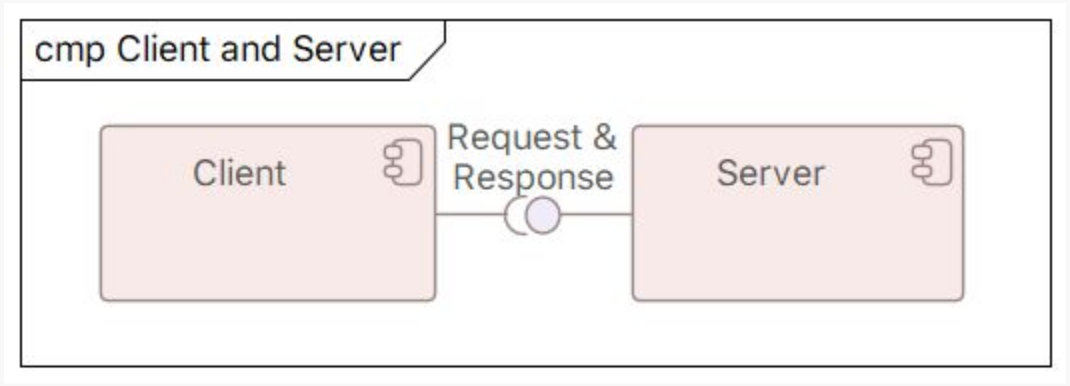
Component Diagram

How to Figure to Determine Provided Interface & Required Interface ?

Client: **Act on** a server via sending a request.

Server: Waits for a request from client to **react**.

Ex (Client & Server):

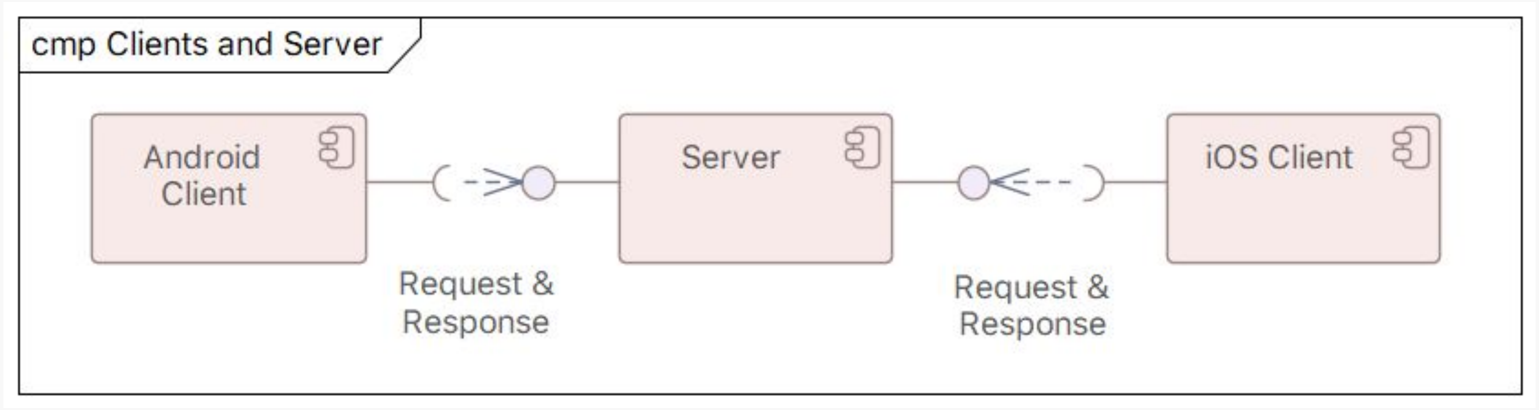


Note: Diagrams /w & /wo dependency lines are interchangeably used as they are equivalent.

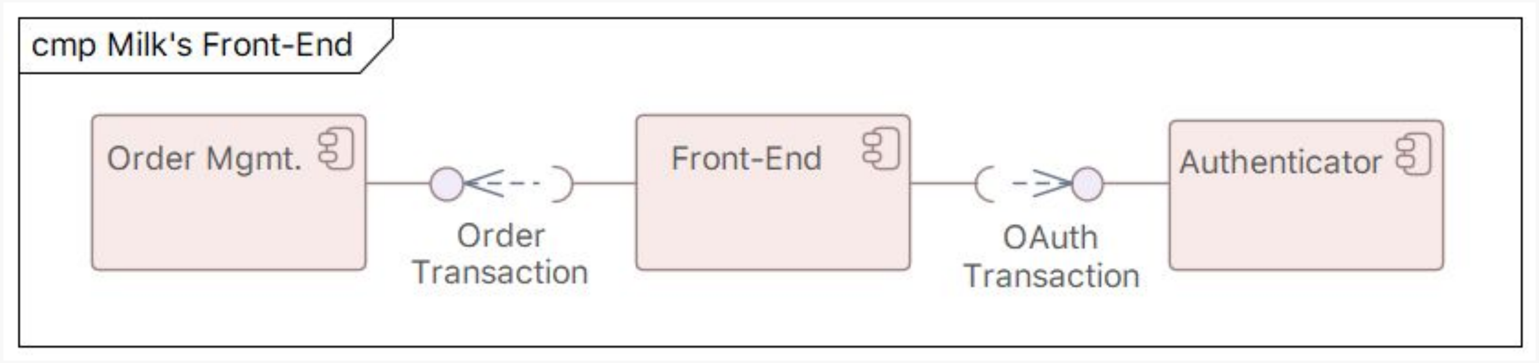
Component Diagram

How to Figure to Determine Provided Interface & Required Interface ?

Ex (2 Types of Client & Server):



Ex (3 Different Services):



Mediator vs Peer-TO-Peer

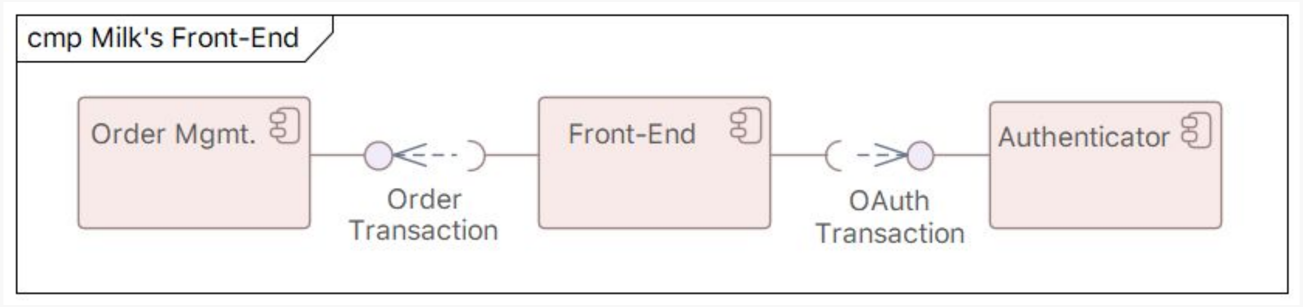
How to Determine A Service As Mediator Or Peer-to-Peer ?

Observe the following aspects in that service**:**

- Use case-based observation = Does that service serves several use cases?
- Component-based observation = Does that service act on more than one service? Does that service react to more than one service?

If answers for **all of the above is no**: Peer-to-Peer

Ex: Front-end (Mediator): 2 Use cases: Authentication + Order Foods
Act on 2 Services: Authenticator & Order Mgmt.



Note: Same Examples in Week 3, but rearranged.

Mediator vs Peer-TO-Peer

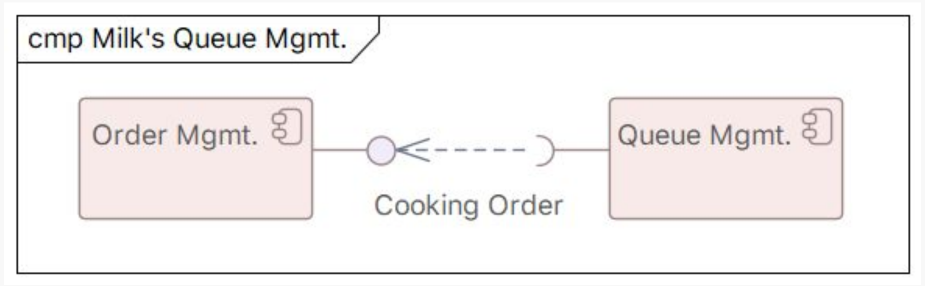
How to Determine A Service As Mediator Or Peer-to-Peer ?

Observe the following aspects in that service**:**

- Use case-based observation = Does that service serves several use cases?
- Component-based observation = Does that service act on more than one service? Does that service react to more than one service?

If answers for **all of the above is no**: Peer-to-Peer

Ex: Order Mgmt (Peer-to-Peer): 1 Use Cases: Manage Cooking Order
Act on 1 Service: Order Mgmt (Get Order Info).



Note: Same Examples in Week 3, but rearranged.

(Not this SOAP)

II: SOAP

23/07 - Web Service Dev & SOA

PROLOGUE

Currently, there is no unified stand for sending messages in SOA.
Two approaches that are widely-used are:

SOAP
(XML-based)

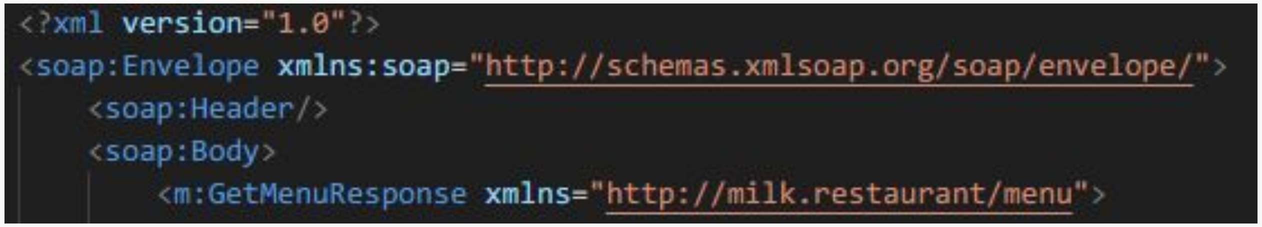
(Cover Today)

REST
(XML and/or JSON)

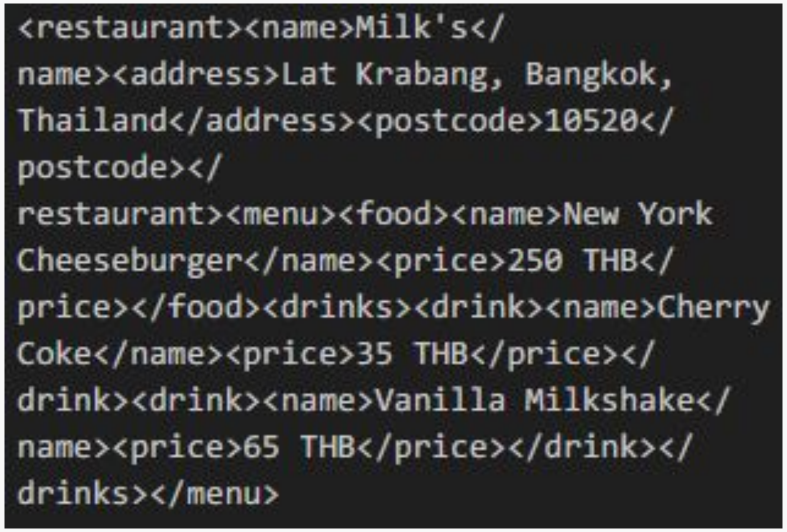
(Covered in Week 2)

SOAP

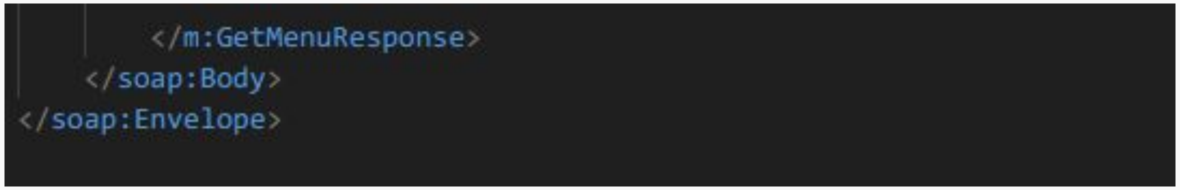
An “Envelope” (Colored Part) for Sending Web Services Messages. XML Based.



```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <m:GetMenuResponse xmlns="http://milk.restaurant/menu">
```



```xml
<restaurant><name>Milk's</
name><address>Lat Krabang, Bangkok,
Thailand</address><postcode>10520</
postcode></
restaurant><menu><food><name>New York
Cheeseburger</name><price>250 THB</
price></food><drinks><drink><name>Cherry
Coke</name><price>35 THB</price></
drink><drink><name>Vanilla Milkshake</
name><price>65 THB</price></drink></
drinks></menu>
```



```xml
        </m:GetMenuResponse>
    </soap:Body>
</soap:Envelope>
```

SOAP

Problem: Why would we use SOAP? Answer: WSDL

Meta-data Sharing

- Can ensure that **all variables has in the right types.**
- Can ensure that **requests are in the right format.**

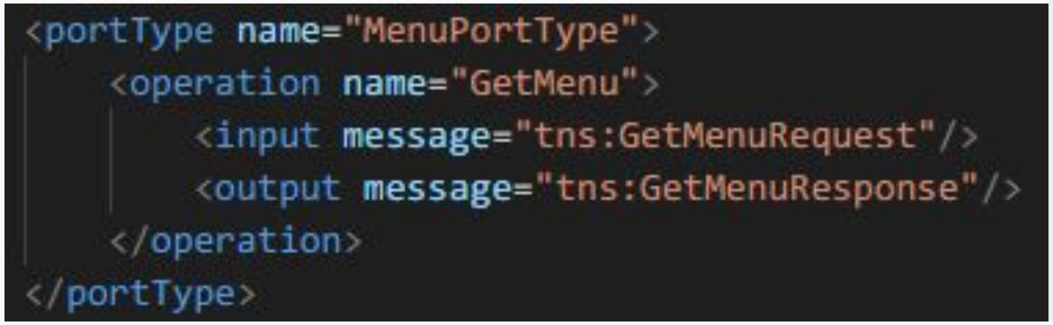
Loosening the coupling between services

- But with its own disadvantage.

WSDL

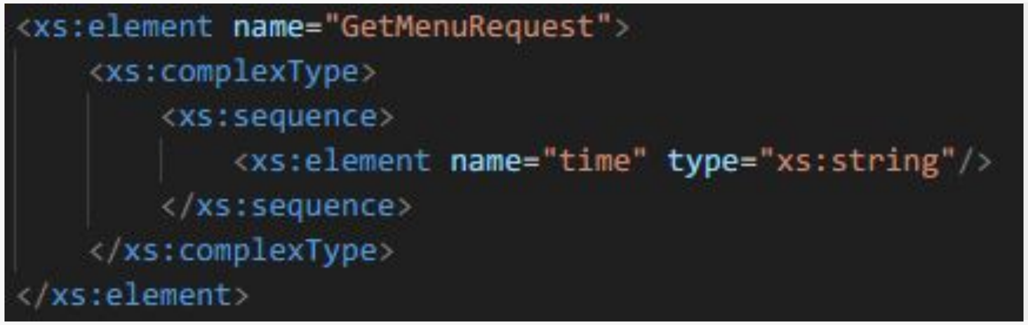
Meta-Data Sharing with Web Services Description Language (WSDL)
Another XML based.

Running Example: Get Menu



```
<portType name="MenuPortType">
    <operation name="GetMenu">
        <input message="tns:GetMenuRequest"/>
        <output message="tns:GetMenuResponse"/>
    </operation>
</portType>
```

Blinding Request & Response from "GetMenu".



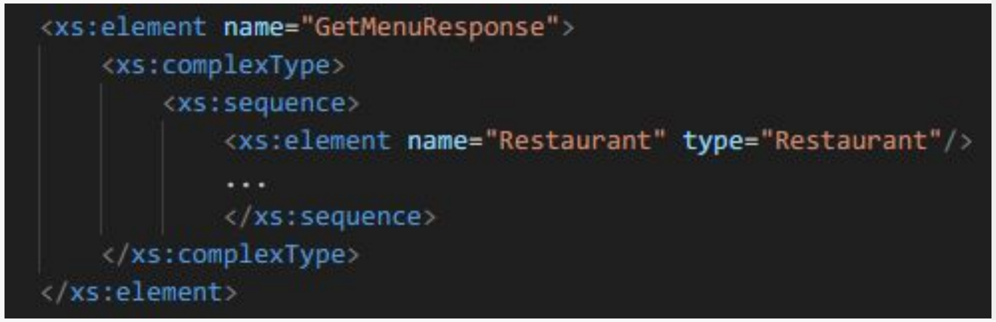
```
<xs:element name="GetMenuRequest">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="time" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Specifying the request to be string named "time".

WSDL

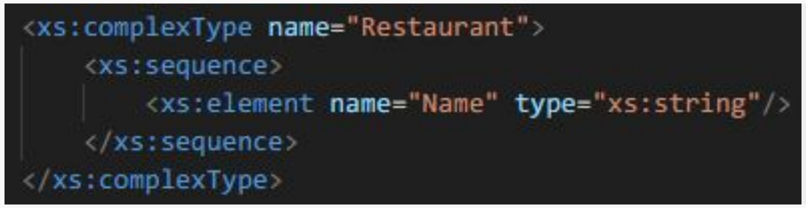
Meta-Data Sharing with Web Services Description Language (WSDL)
Another XML based.

Running Example: Get Menu



```
<xs:element name="GetMenuResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Restaurant" type="Restaurant"/>
            ...
            </xs:sequence>
    </xs:complexType>
</xs:element>
```

Specifying the response to return a menu with “Restaurant”.



```
<xs:complexType name="Restaurant">
    <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

… Where “Restaurant” consists of “name” which is string.

Premise: If all involved services get the same WSDL = Unified data format & requests.

WSDL

Introducing three different components that required to make all involved services have the same WSDL :

SP: Service Provider

B: Broker/ Registry

SR: Service Requester

WSDL

These three components need to commit in the three following phase to ensure the same WSDL:

Declaration

1. SP **declares** WSDL to B.

Brokering

2. SR send a WSDL request to **B.**
3. **B respond with** WSDL to SR.

Communication

4. SR send a request to SP.
5. SP respond to SR.

WSDL

Real-life Example: AirBnB (/w Tenant's Screening Process)

Declaration

1. Tenant **declares** room availability to AirBnB.

Brokering

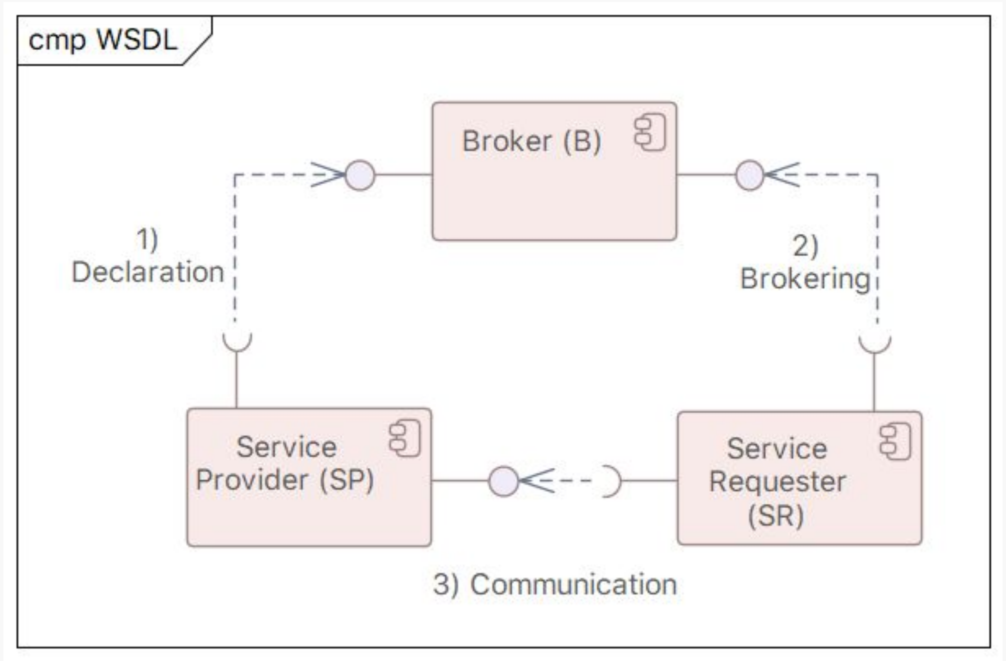
2. Client send a booking request **to AirBnB**
3. **AirBnB respond with** Tenant's contact info to Client.

Communication

4. Client contact the Tenant for reservation.
5. Tenant screen & respond to Client's reservation.

WSDL

Component Diagram:



1) Declaration

- SP **declares** WSDL to B

2) Brokering

- SR send a WSDL request to **B**
- **B respond with** WSDL to SR

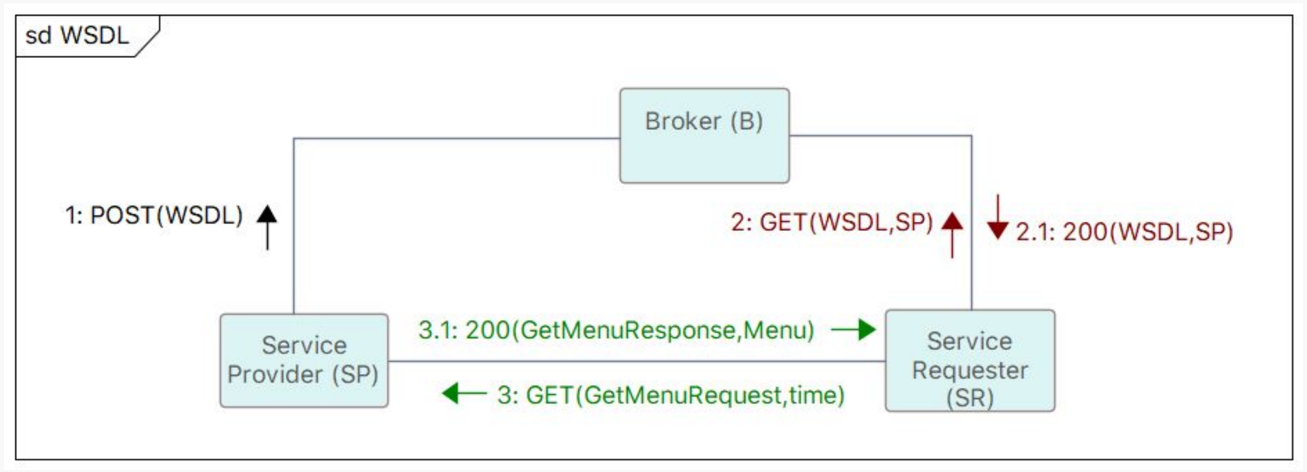
3) Communication

- SR send a request to SP
- SP respond to SR

WSDL

Communication Diagram (assuming all requests/responses are synchronous):

(Note: Colored arrows = Synchronous reqs.)



Declaration

1. SP **declares** WSDL to B

Brokering

2. SR send a WSDL request to **B**

2.1. B respond with WSDL to SR

Communication

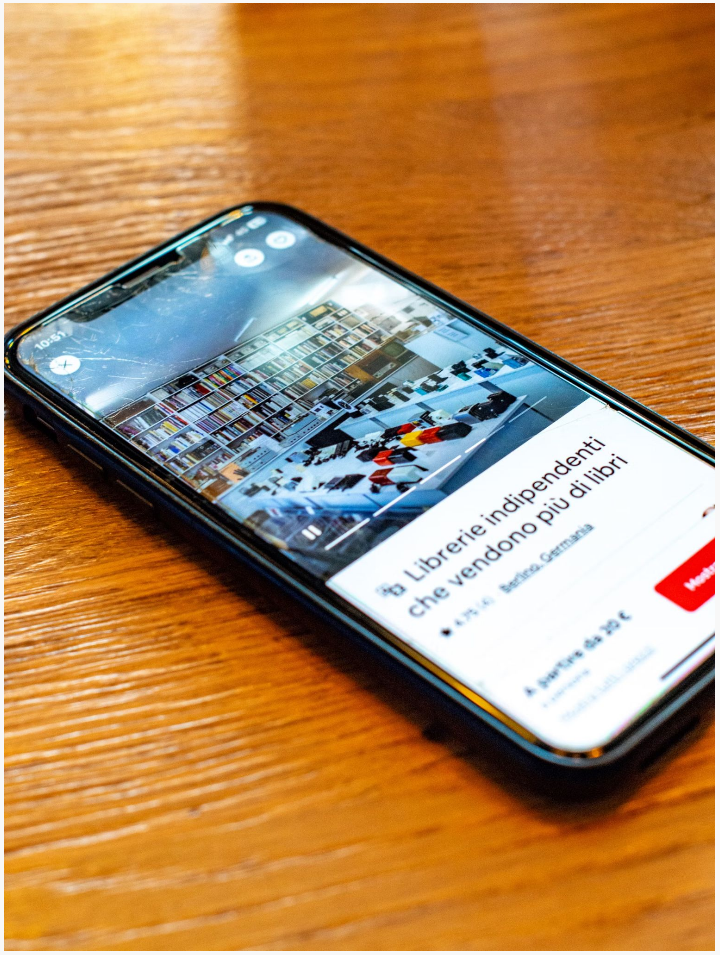
3. SR send a request to SP

3.1. SP respond to SR

WSDL

Advantages

- **Separation of concern:** Loosen **the coupling** between SP & SR: They **do not need to be hard-coded** from the beginning. SP & SR can be independently developed. Any update? **Just ask B**.
- **Correctness**: Ensure all variables and requests are correct (via WSDL). If not, **just update to what B sent**.

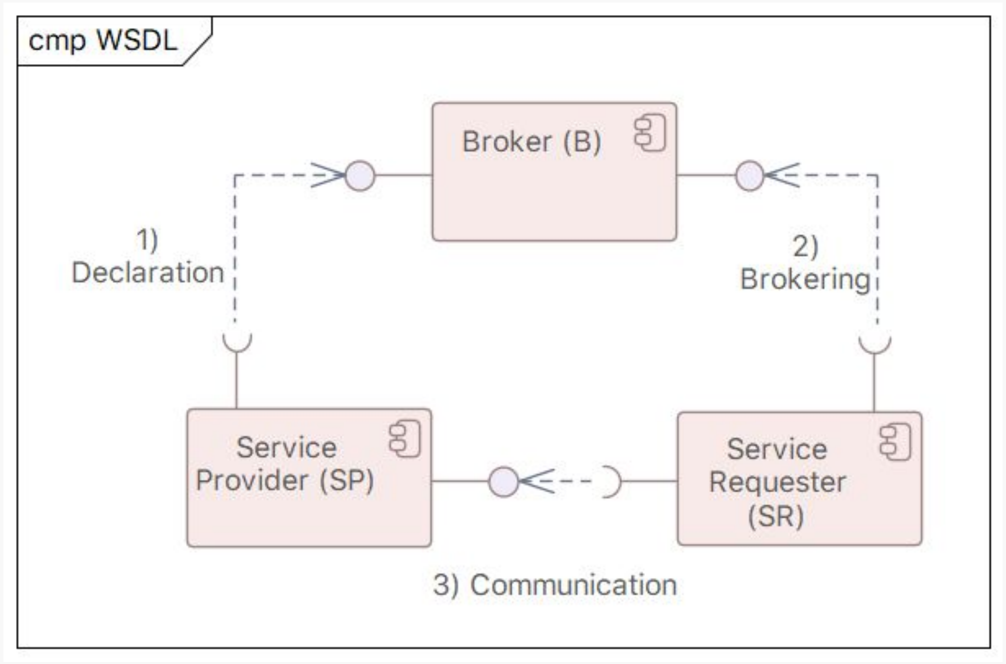


WSDL

Disadvantages

B is a **mediator** (2 Use cases/phases, interact to 2 services). All mediator disadvantages apply, including:

- **High Impact from the attack: If B got hacked,** know all requests/responses to SP. **WSDL from SR can be altered**.
- **Single Point of Failure: If B downed,** a new SR cannot connect to SP until B is recovered.



III: ESB

23/07 - Web Service Dev & SOA

Prologue

Currently, there is no unified stand for sending messages in SOA.
Two approaches that are widely-used are:

SOAP
(XML-based)

(Cover in II)

REST
(XML and/or JSON)

(Covered in Week 2)

Prologue

Since there is no standardised approach for SOA, it is possible to use both of them.
Problem: How to use/switch between them without “heavy-coding”?

SOAP	**REST**
(XML-based)	*(XML and/or JSON)*
(Cover in II)	**(Covered in Week 2)**

ESB

Introducing Enterprise Service Bus. A middleware for communication within a SOA system. Key functions include:

Translator

REST to SOAP? SOAP to REST? Just let me know.

Transformer

JSON to XML? XML to JSON? Got it.

Monitor

Any service got too much traffic? See it here.

Security

SSL? HTTPS? Encrypt/ decrypt here.

ESB

Introducing Enterprise Service Bus. A middleware for communication within a SOA system. Real-life example: Shipping Services.

Translator

International to National? National to International? We covered them all.

Transformer

Combined shipping? New packaging? We got it.

Monitor

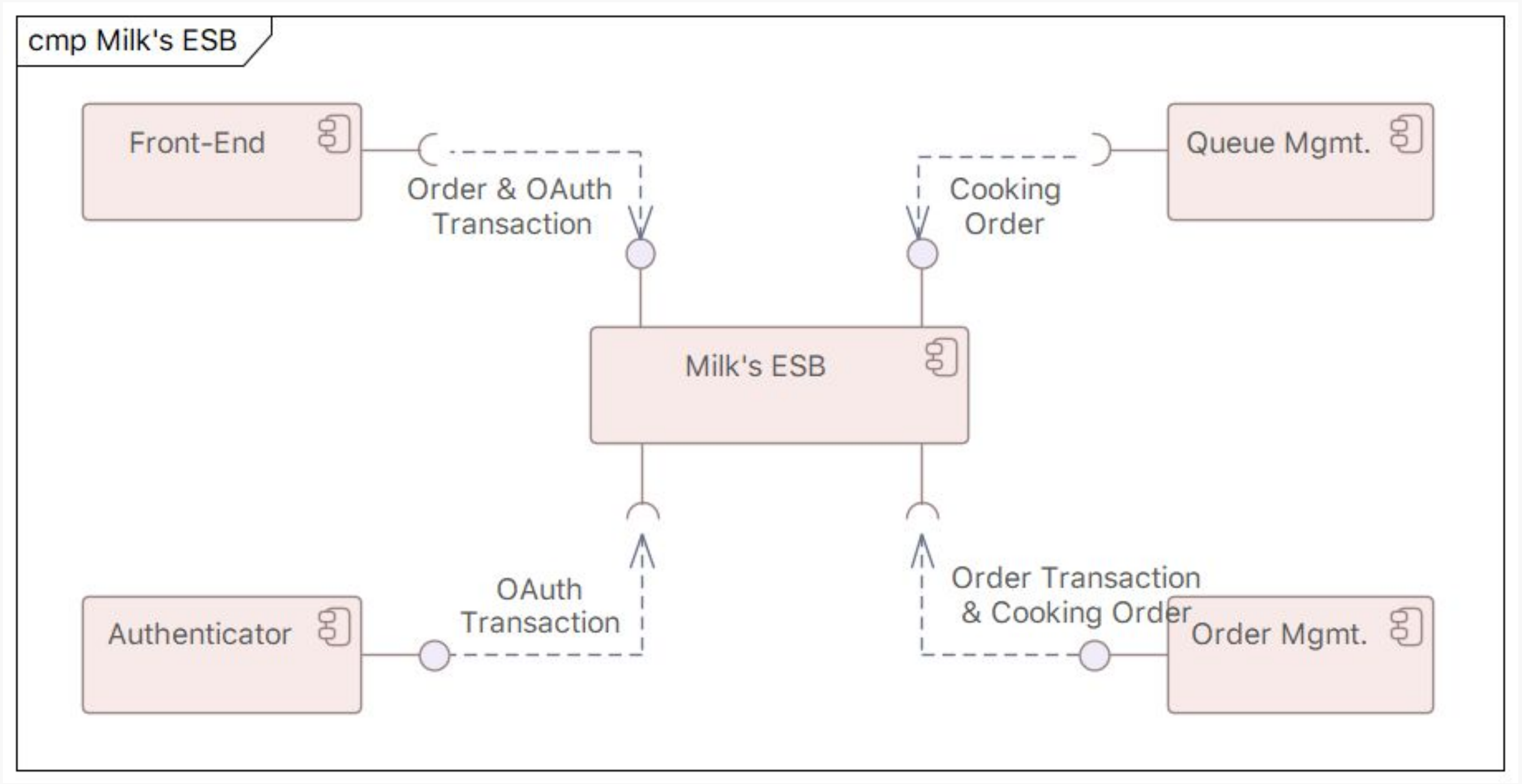
Your shipment? Track with Tracking ID.

Security

Import Tax? Security Clearance? We got you.

ESB

A running example - Milk’s Restaurant ESB (Component Diagram):

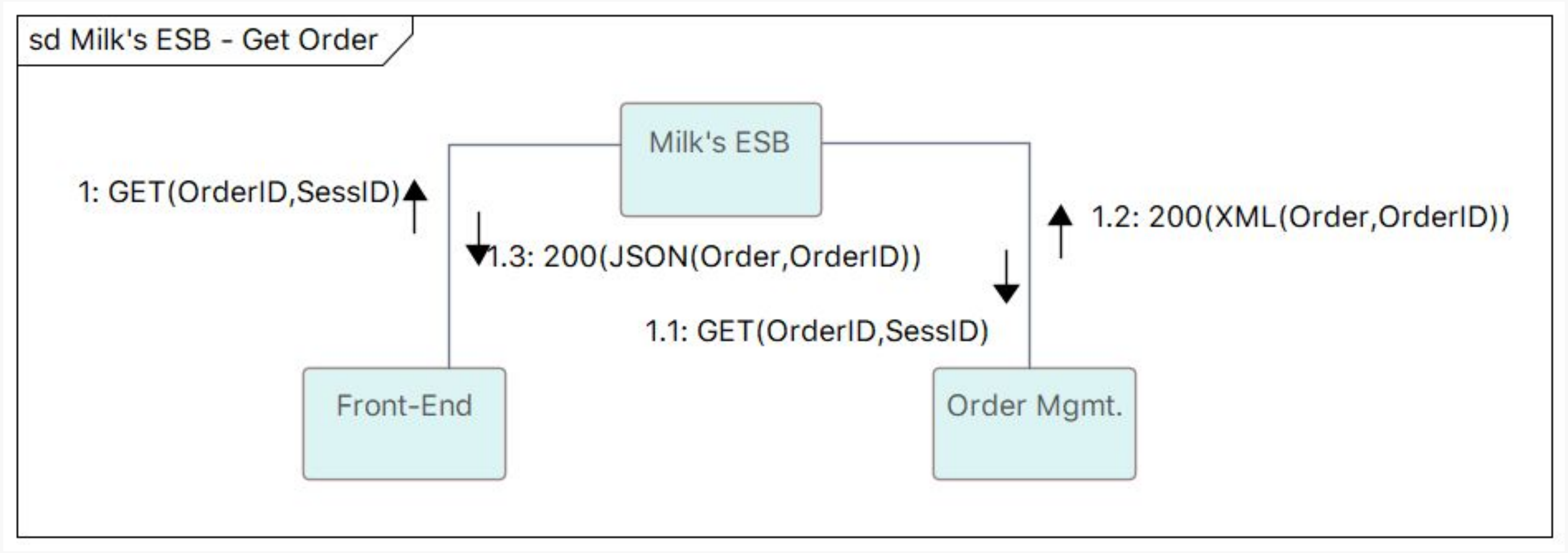


Observation: ESB is a Mediator.

ESB

A running example - Get an order (Communication Diagram):

(Note: Notice 1.2 & 1.3)


sd Milk's ESB - Get Order
Milk's ESB
1: GET(OrderID,SessID)
1.3: 200(JSON(Order,OrderID))
1.2: 200(XML(Order,OrderID))
1.1: GET(OrderID,SessID)
Front-End
Order Mgmt.

(Note II: Colored arrows = Synchronous reqs.)

ESB

Advantages:

Interoper-ability

Can handle any messaging protocols (as long as ESB capable).

Convenience

Reduce complexity in develop & integrate new services.

Trace-ability

Capable to track/trace traffic in each service.

Unified Security Standard

Centralised security mechanism ensure that all services are **equally secured.**

ESB

Disadvantages - Similar to any Mediator service:

Heavy weight

Increase **complexity** to develop, maintain and evolve.

Bottleneck

Increase **overhead** in messaging between services.

Single Point of Failure

If ESB downs, **the system fails**.

Unified Security Standard

Centralised security mechanism means the same vulnerability **applies to the whole system.**

Future Starts Slow

Although WSDL/ESB have disadvantages, they may be necessary for:

Legacy System

Packaging the legacy system for ESB/WSDL can be less expensive than building a new system.

Large System

Creating ESB/WSDL can be less expensive than making all services to conform to one or new standard.

Existing System with WSDL/ESB

Costly or infeasible to replace them **(at this point).**

Security Trade-off

Centralised standard/sec urity mechanism across the system is still **better than nothing.**

Group Exercise

23/07 - Web Service Dev & SOA

Group Exercise -Week 4

1. Do you think REST or SOAP or both is necessary for your system? Rationalise why it is the case.
2. Elaborate a scenario where ESB is essential for your system. Illustrate how the ESB can be used in your system (via a communication diagram).

Send To:
suwichak.fu(at)kmitl.ac.th

Subject:
[6622][(Team Name)][IoT/Metaverse] Group Exercise Submission

Example:
[6622][Nanno][Metaverse] Group Exercise Submission