**SOA and SOAP and Implementation**

**Problem**

* **History=> Old time late 80's functional programming:**

Top to down approach as waterfall model.

Eg:

function (input1, input2 )//entry point

{

input1=Console.ReadLine();

input2=Console.ReadLine();

Console.WriteLine(Add(input1+input2));

}

//-----------Wrapping this function

function Add(int a, int b)

{

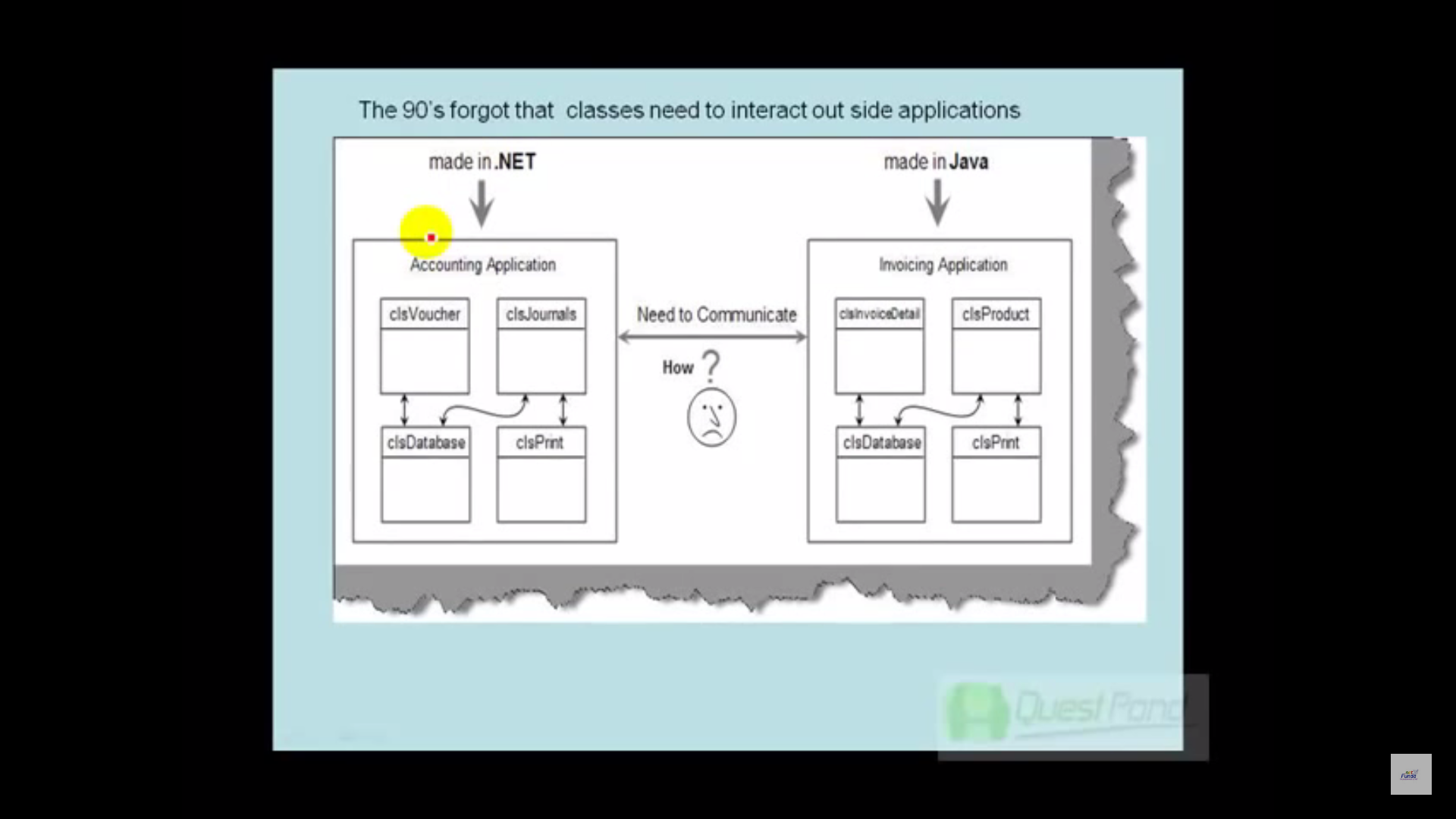
return a+b;

}

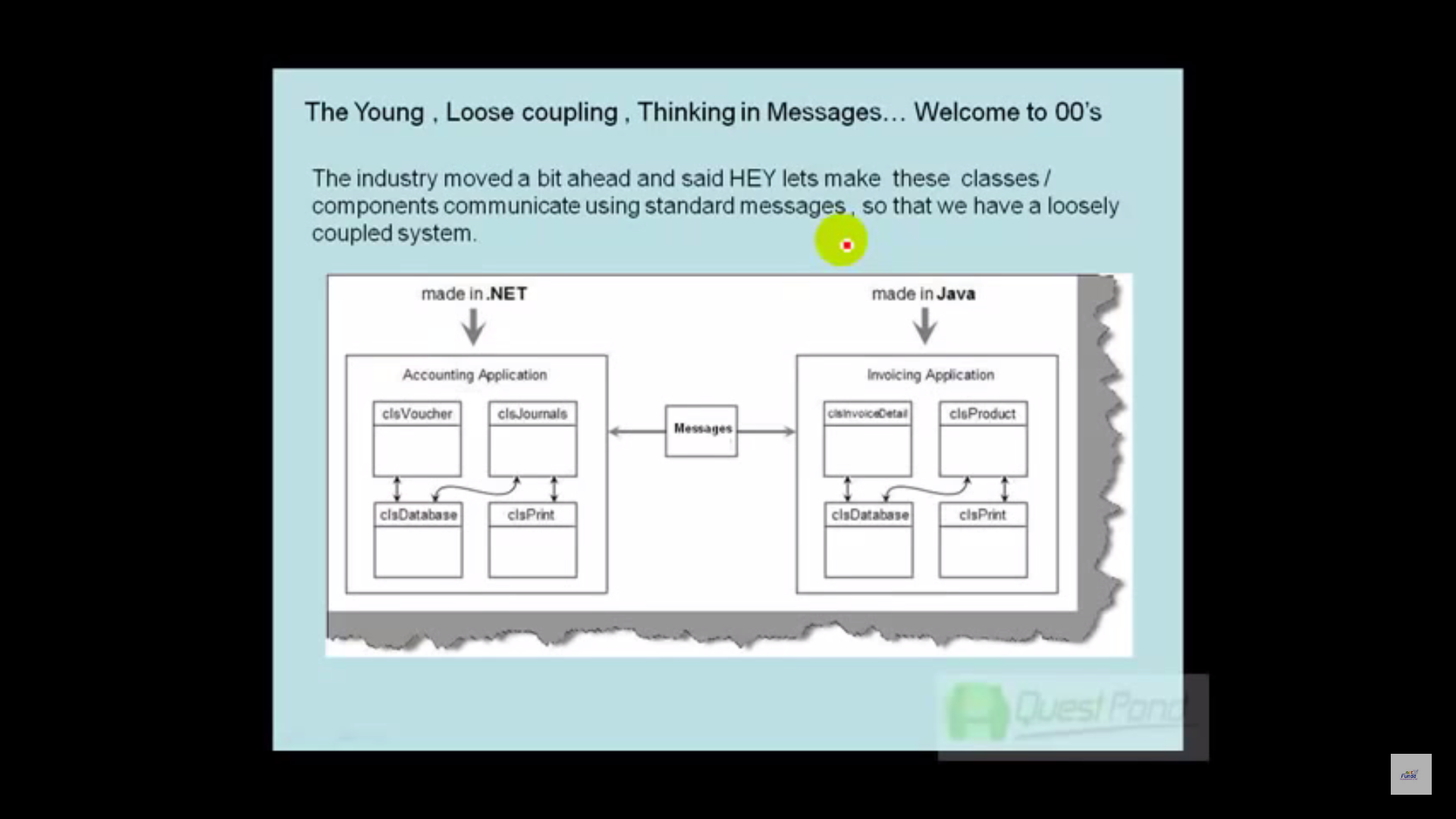
* **Old time late 90's functional programming:**

Came Classes with OOPS concepts for reusability and encapsulating

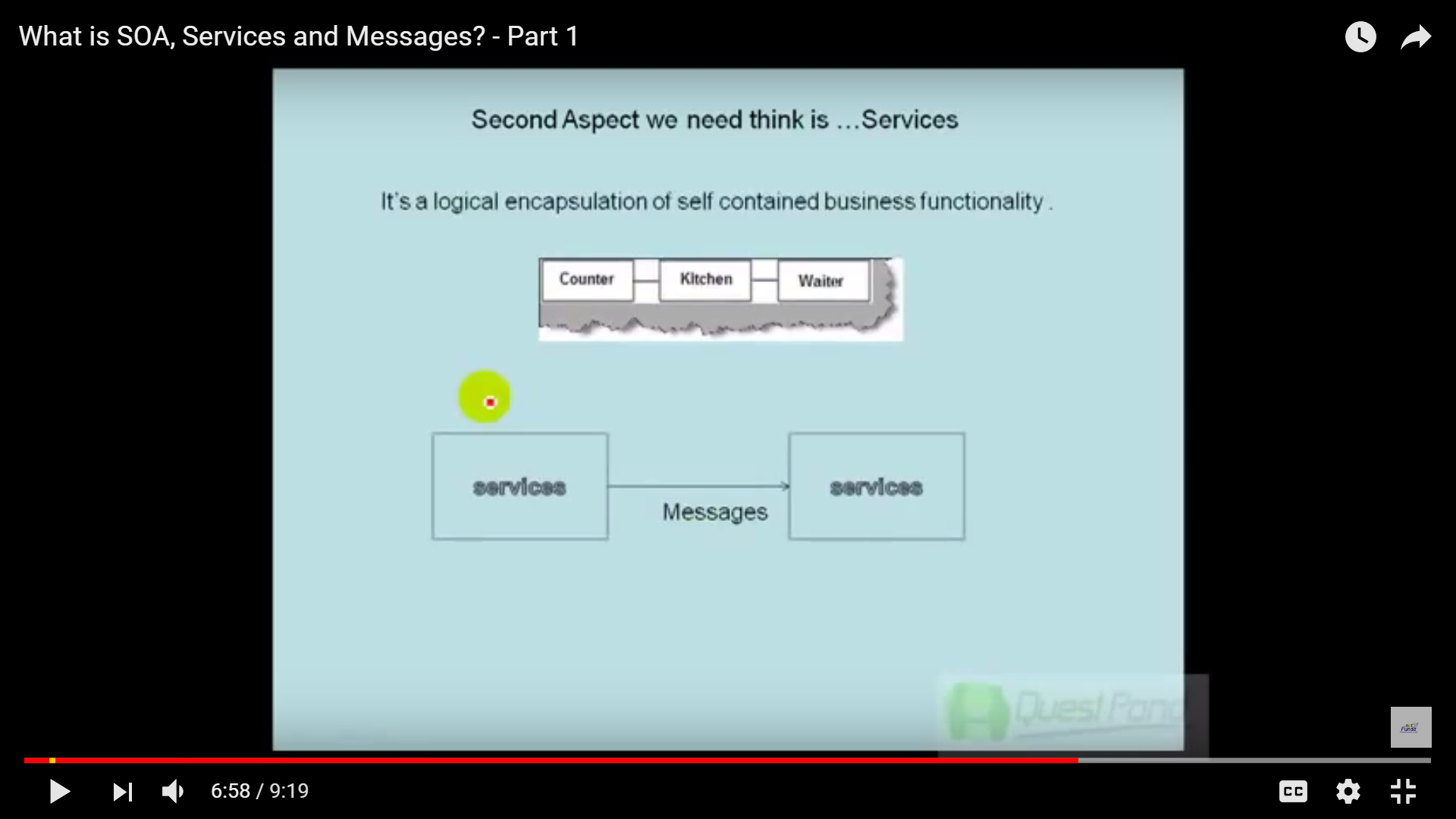
**Problem** :



Solution



No need to think of proprietary programming compilers and only processing standard messages.



**Services**

It's a logical encapsulation of self-contained business functionality". What does that mean? It means that the service is (each and every service) logically encapsulated within some other container (WCF server) and those services are self-contained.

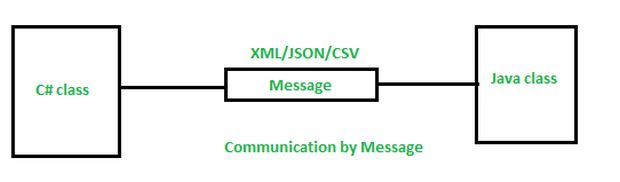
Services are not classes or namespaces or objects. It's just self-contained business functionality.

**Message**:

how two applications will talk

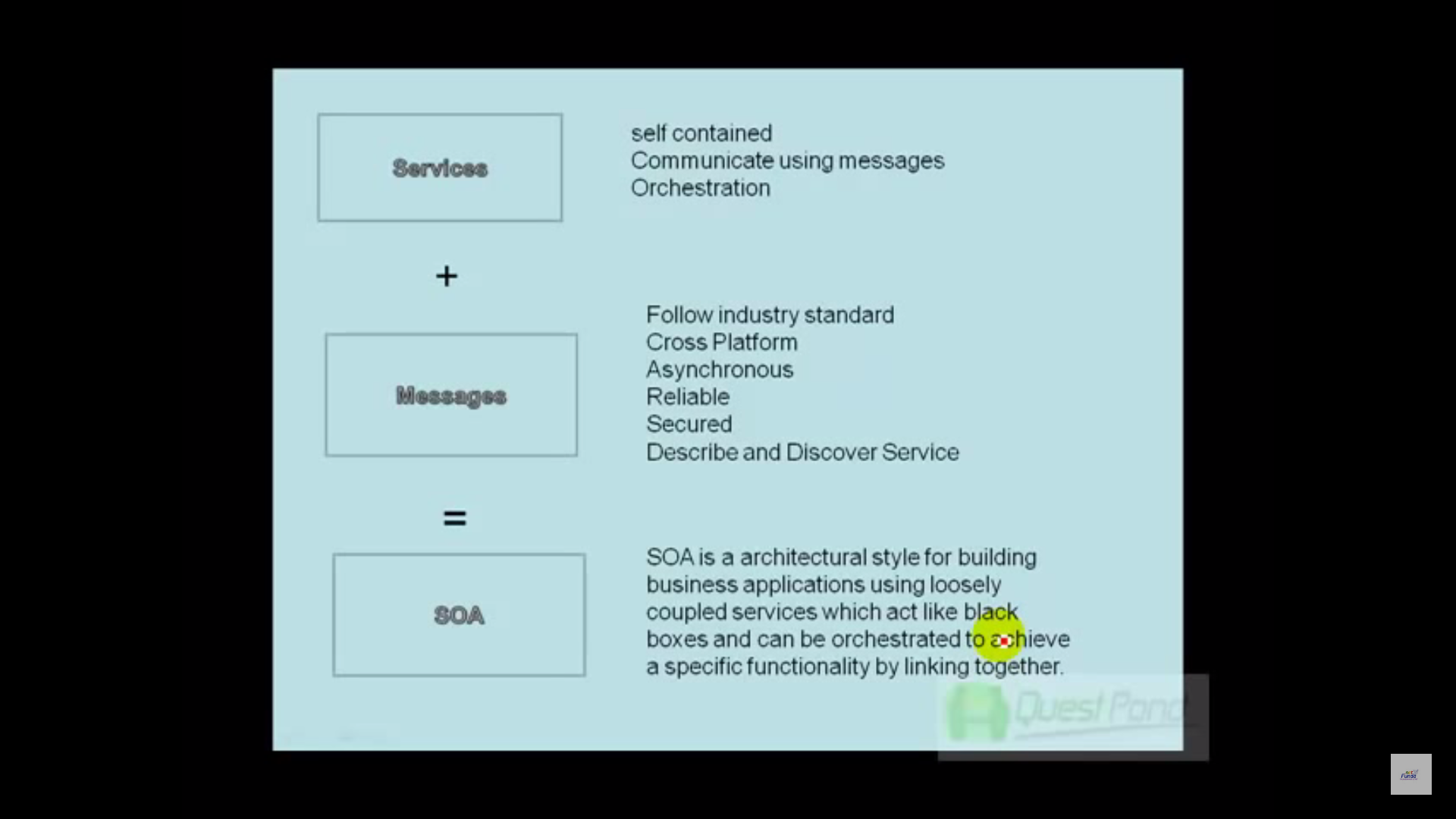
The format can be XML, JSON or even a plain CSV file. Here is a simple diagram to understand the concept of messages.

*Help Link*  
<http://www.c-sharpcorner.com/UploadFile/dacca2/understand-wcf-part-1-what-is-soa-services-and-messages/>



Thumb Rules for Services and Messages:

1. Services should be self-contained and should use messages to communicate.
2. Services should be capable of self-defining themselves or self-explanatory like mentioning the functionality it has.
3. Services should be maintained in a central repository where Application consumers can find and use them
4. Services should be Orchestrated- Should allowed to define the work flow. Eg: todo Security check first and processing user request later based on inputs or vice versa.
5. Messages should be standard in format and cross-platform compatible. Eg Json, XML, CSV.
6. Message should be communicated asynchronously, securely and reliably.
7. Message should be able to describe and discover a service in a standard manner. Like protocol to be used.



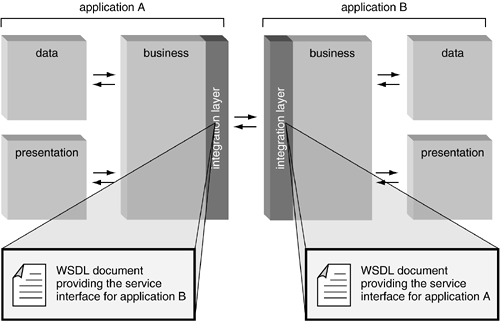
**What is SOA?**

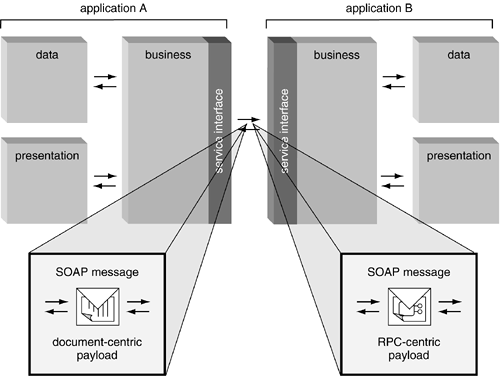
The Service Oriented Architecture is an architectural design which includes collection of services in a network which communicate with each other

 SOA is nothing but an architectural style where two non-compatible applications/systems can communicate using a common language and WCF is nothing but one example of the applications service Oriented Architecture (SOA).

Advantages

* SOA allows reuse the service of an existing system alternately building the new system.
* It allows plugging in new services or upgrading existing services to place the new business requirements.
* It can enhance the performance, functionality of a service and easily makes the system upgrade.
* SOA has capability to adjust or modify the different external environments and large applications can be managed easily.
* The companies can develop applications without replacing the existing applications.
* It provides reliable applications in which you can test and debug the independent services easily as compared to large number of code.





**Services in .Net Framework:**

1. **SOAP Service**
   1. WebServices: http(s) protocol
   2. WCF SOAP Service: support of multiple protocols [http, tcp, namedPipes, MSMQ]

Message: XML

Protocol: SOAP

WSDL : <http://www.oracle.com/technetwork/java/understanding-wsdl-138582.html>

**WCF Contracts:**

* + - ServiceContract
    - OperationContract
    - FaultContract
    - MessageContract
    - DataContract

**Service Contract**

Service contract **describes the operations**, or methods, **that are available on the service endpoint**, and exposed to the outside world. A Service contract describes the client-callable operations (functions) exposed by the service, apart from that it also describes the operation contract.

**OperationContract**

Methods in the interface that should be included in the service contract are decorated with the ***OperationContract***Attribute.

**Data Contract**

In one line, data contract **describes the data to be exchanged**. it is formal agreement between service and client, that **contains information about the data they will be exchanging**. the important point, which needs to be mentioned here is that the two parties don’t have to share the same data types to communicate, they only need the share the same data contracts.

 Data contracts can be defined by annotating a class, enumeration, or even a structure, but not an interface.

**Message Contract**

WCF uses SOAP message for communication. Most of the time developer concentrates more on developing the **DataContract**, Serializing the data, etc. Some time developer will also require control over the SOAP message format. In that case WCF provides Message Contract to customize the message as per requirement.

A **Message Contract** is used to control the structure of a message body and serialization process. It is also used to send / access information in SOAP headers. By default WCF takes care of creating SOAP messages   
according to service **DataContracts** and **OperationContracts**.



So when you need full control on SOAP messages structure and how serialization happens specially when your service needs to be interoperable for consuming by different types of clients or service needs to provide extra layer of security on messages and message parts, A message is nothing but a packet and WCF uses this packet to transfer information from source to destination. This message contains an **envelope**, **header**and **body**. There are some rules, which one needs to follow, while working with  message contract.

* When using Message contract type as parameter, Only **one parameter can be used in Operation**.
* Service operation either **should return MessageContract** type **or** it **should not return any value**.
* Operation will **accept and return only message contract type**. Other data types are not allowed

**Choosing between DataContract and MessageContract.**

**90%** of the time, **DataContract will be sufficient**, You'll only need message contracts if you need to very closely and very specifically control the layout of your SOAP messages. **In most of the time,  you don't need to**.

A message contract allows you to specifically say which elements (scalar types or compound types as **DataContracts**) will be in the SOAP header, and which will be in the SOAP body.

You might need this if you have a communication partner, with whom you have agreed to a very specific format and you have to tweak your SOAP messages to match that given layout exactly. That's just about the **only valid scenario**when **you'll need to** and **should use message contracts**.

However, sometimes complete control over the structure of a SOAP message is just as important as control over its contents. This is especially true when **interoperability is important or to specifically control security issues at the level of the message or message part**. In these cases, you can create a *message contract* that enables you to use a type for a parameter or return value **that serializes directly into the precise SOAP message that you need**.

So, to making the long story short: **always** use **data contracts**, practically never use message contracts (**unless you really have to**).

**Fault Contract**

Fault Contract **provides documented view for error accorded in the service to client**. This help as to easy identity the what error has occurred, and where. By default when we throw any exception from service, it will not reach the client side. The less the client knows about what happened on the server side, the more dissociated the interaction will be, this phenomenon (not allowing the actual cause of error to reach client). is known as **error masking**. By default all exceptions thrown on the service side always reach the client as **FaultException,** as by having all service exceptions indistinguishable from one another, WCF decouples the client from service.

**Revision**

Web Services:

1. SOAP: Simple Object Accesss Protocol

Medium:SOAP Protocol =>transport: http, https, tcp, smtp, msmq, namedpipes,udp

POST

Format: XML

SOAP Message: Envelope\*

Header=> authorisation

Body\*=> actual data sent or recieve

Consumer:URI

What are services available ?

UDDI=> universal Description, Discovery Integration

provider publishes the service=> Online directory/registry/repository

WSDL=> Web Service Description Language

XML based interface

SOAP Services: web service which follows SOAP web services specifications

=> principles of services and message

Specification:

SOAP

WSDL

UDDI

WDSL:

Service

Port

Address

Binding

Protocol

Operations

PortType

Operations

Message

Input=> Types

Output=> Types

Types

WCF: Contracts=> ServicesContract, Operations, FaultContract, MessageContract,

DataContract

Endpoints: A, B, C