Avancier Methods (AM)

Software Architecture

Decoupling - part 1

(LPC, RPC, DO, SOAP, WS)

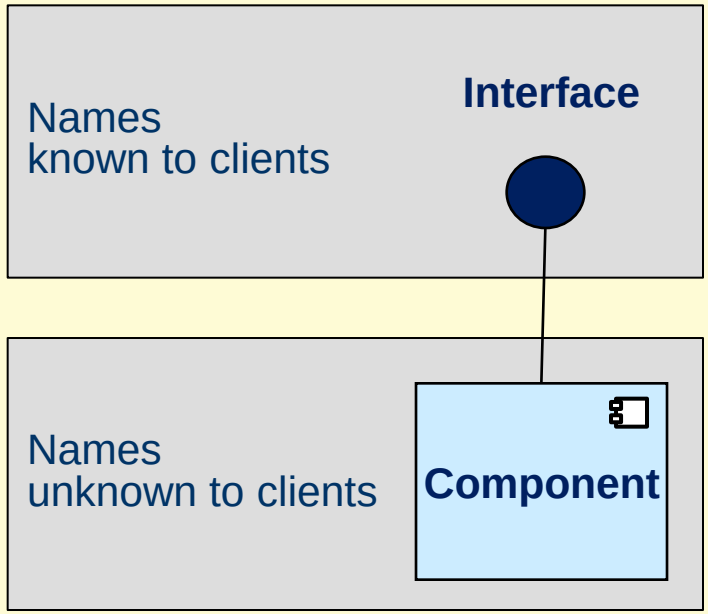
Principles example 3 – a global organisation

1. Separate concerns (for flexibility and maintainability)
2. Build for competitive advantage / Buy for competitive parity
3. Encapsulate components (for CBD and SOA)
4. Use open APIs for inter-component communication
5. Loosely couple components (for flexibility and availability)
6. Use Event-Driven Architecture for broadcast updates
7. Maintain a single source of truth
8. Design for response time / latency
9. Design for graceful failure – informing users
10. Web first: design for browser and client device independence

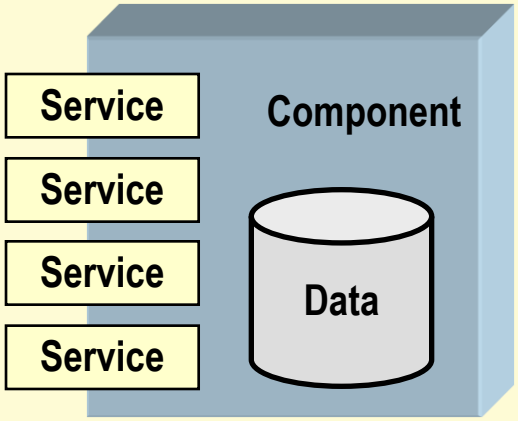
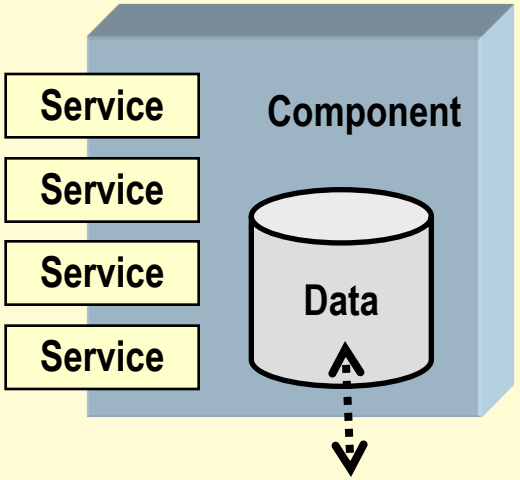
Clients use domain names

Feature	Early OO design presumptions	Recent SOA design presumptions
Naming	Clients use object identifiers Remote objects are located using rather Object Request Brokers	**Clients use domain names** Remote modules are located using Domain Name Servers

Multiple name spaces *(behind interfaces)*

Feature	Early OO design presumptions	Recent SOA design presumptions
Naming	One name space	**Multiple name spaces behind interfaces** 

Stateless objects /modules

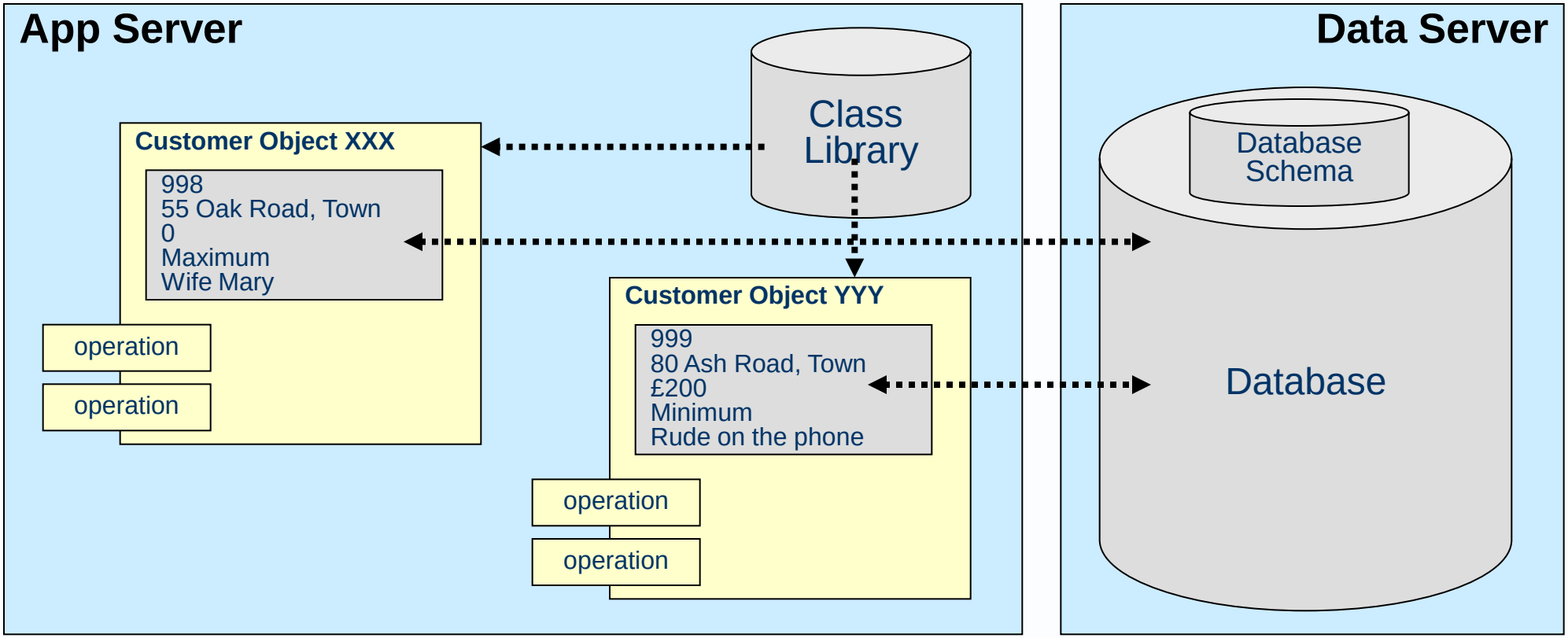
Feature	Early OO design presumptions	Recent SOA design presumptions
Paradigm	Stateful objects/modules Retain state between processes 	**Stateless objects/modules** Retain state only long enough to complete a process 

Objects are transient and stateless (rather than stateful)

Avancier

OOPer view
Objects model real world entities, are stateful and persist.
Databases are infrastructure devices; SQL is evil.

DBA view
Database records model real world entities. App server objects should be stateless and hold data only while processing it

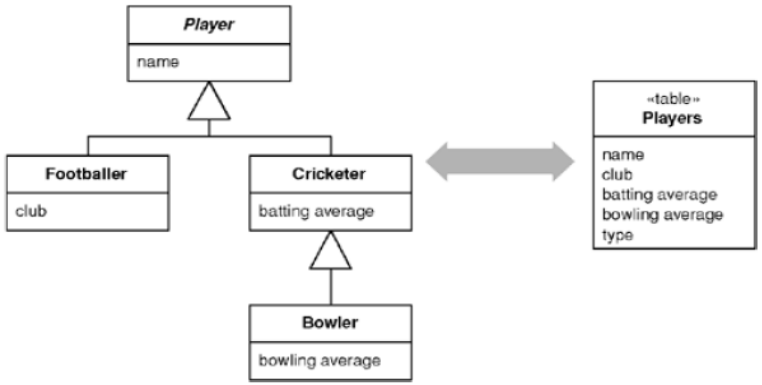


Mapping OO class hierarchy to Database tables (Fowler)

Avancier

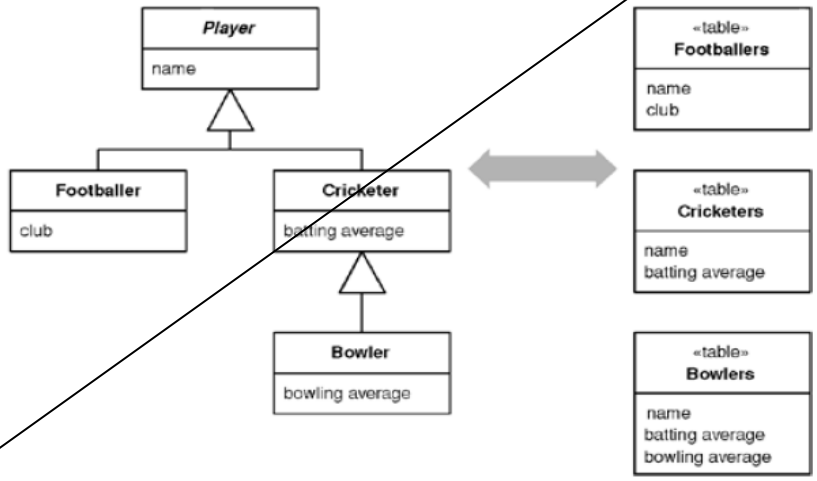
Single Table Inheritance

Represents an inheritance hierarchy of classes as a single table that has columns for all the fields of the various classes.



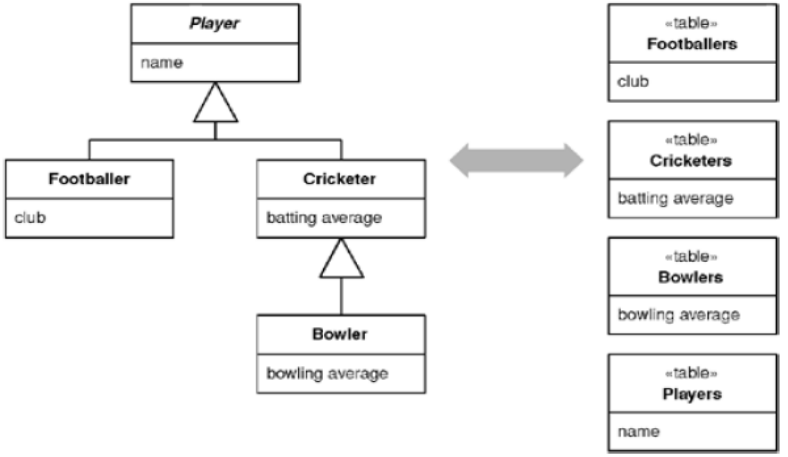
Concrete Table Inheritance

Represents an inheritance hierarchy of classes with one table per concrete class in the hierarchy.

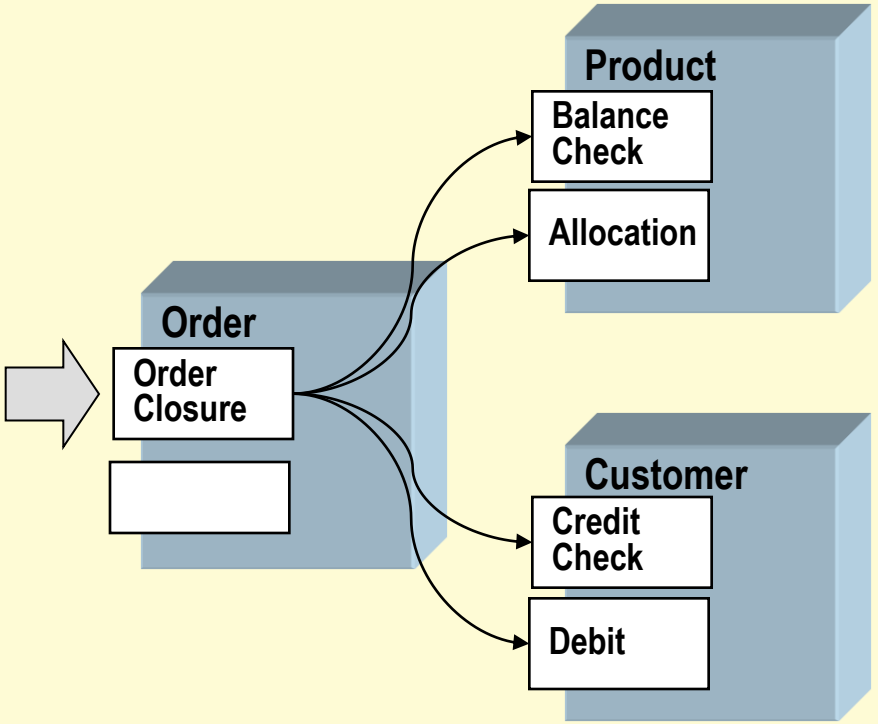


Class Table Inheritance

Represents an inheritance hierarchy of classes with one table for each class.



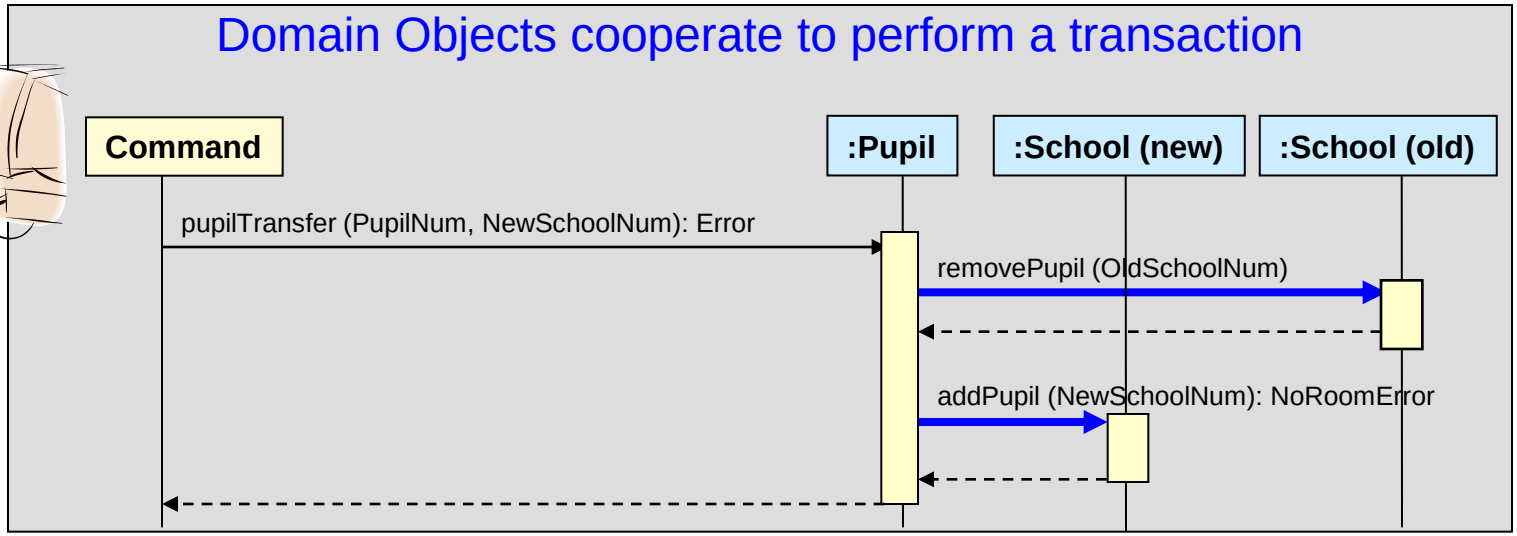
Reuse by delegation / invocation

Feature	Early OO design presumptions	Recent SOA design presumptions
Paradigm	Reuse by OO inheritance	**Reuse by delegation** The traditional reuse mechanism 

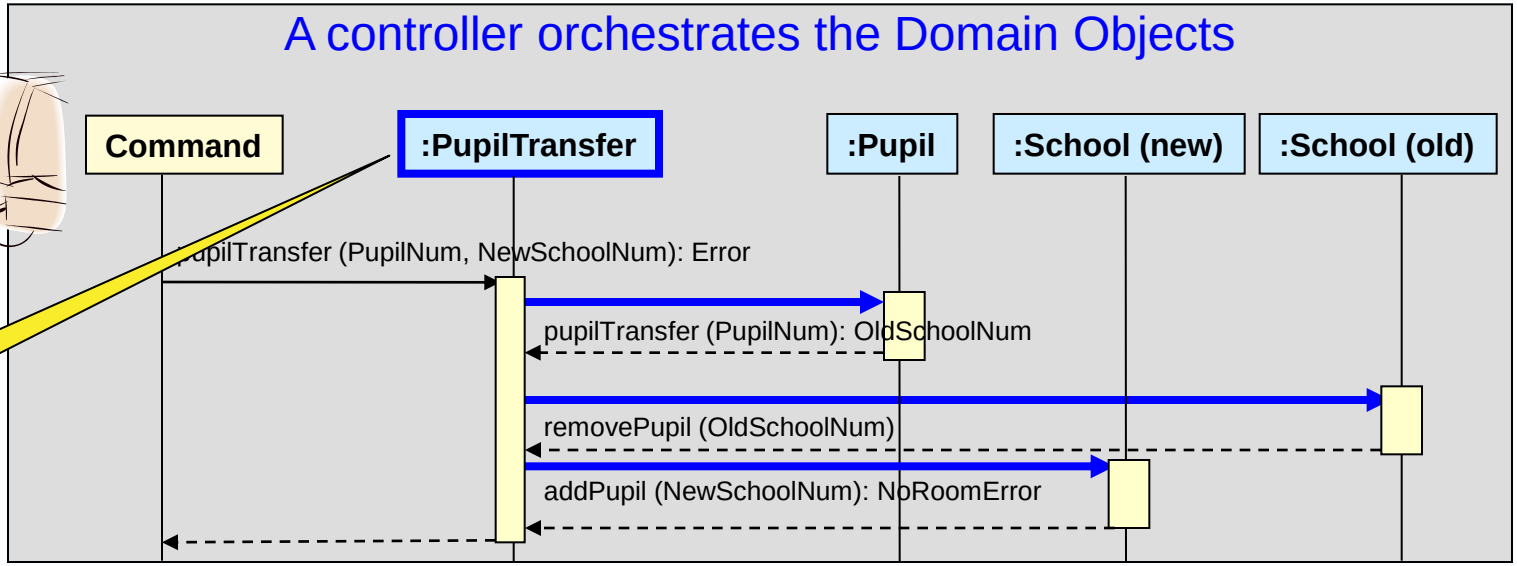
Intelligent process controllers (hidden in aggregate root entities?)

Avancier

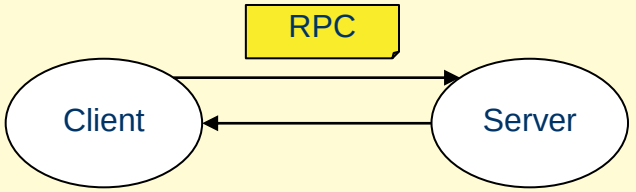
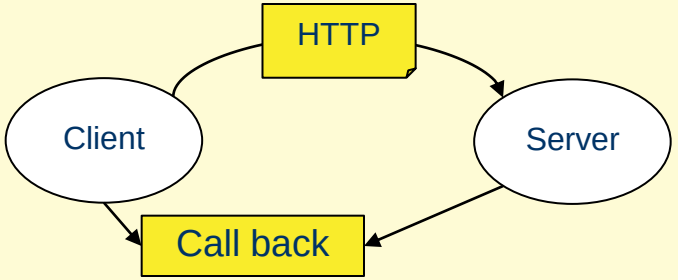
Chain/ Choreography



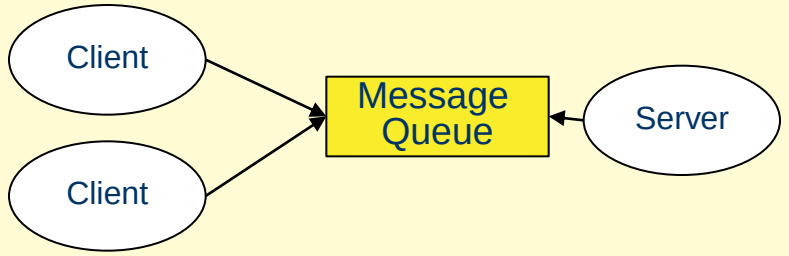
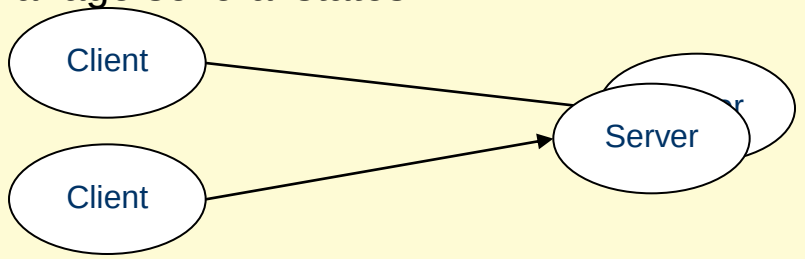
Fork/ Orchestration



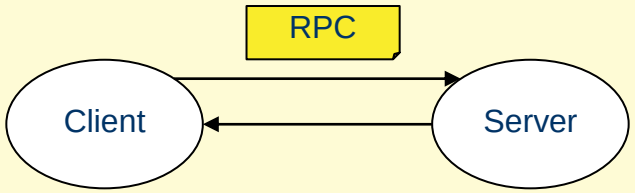
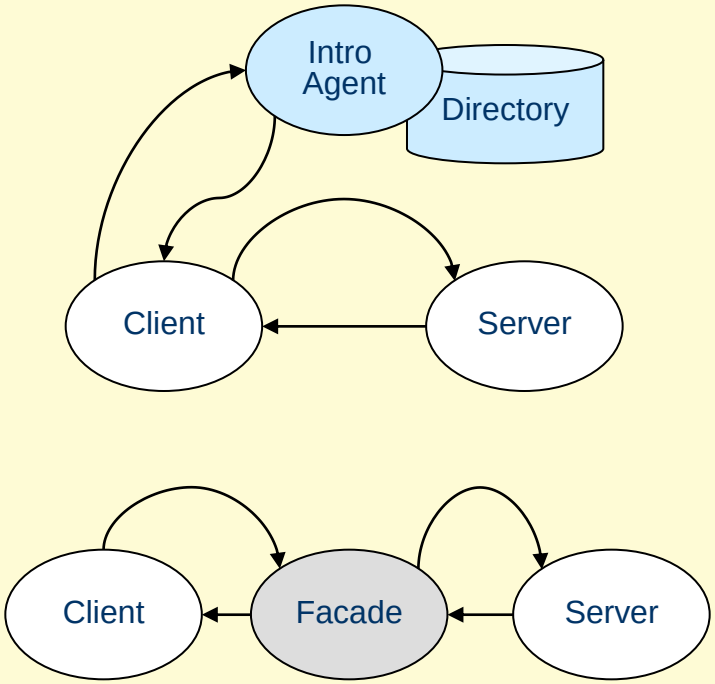
Decoupling from time/availability

Feature	Early OO design presumptions	Recent SOA design presumptions
Time	Request-reply invocations 	**Asynchronous messaging** A client can use a call back mechanism, or subscribe to be notified of a reply event 

Decoupling from time/availability

Feature	Early OO design presumptions	Recent SOA design presumptions
Time	Blocking servers	**Non-blocking servers** Client invocation messages are held in a message queue  Or the server is multi-threaded, so can manage several states 

Decoupling from location

Feature	Tight coupling	Decoupling techniques
Location	Remember remote addresses 	**Use brokers/directories/facades** 

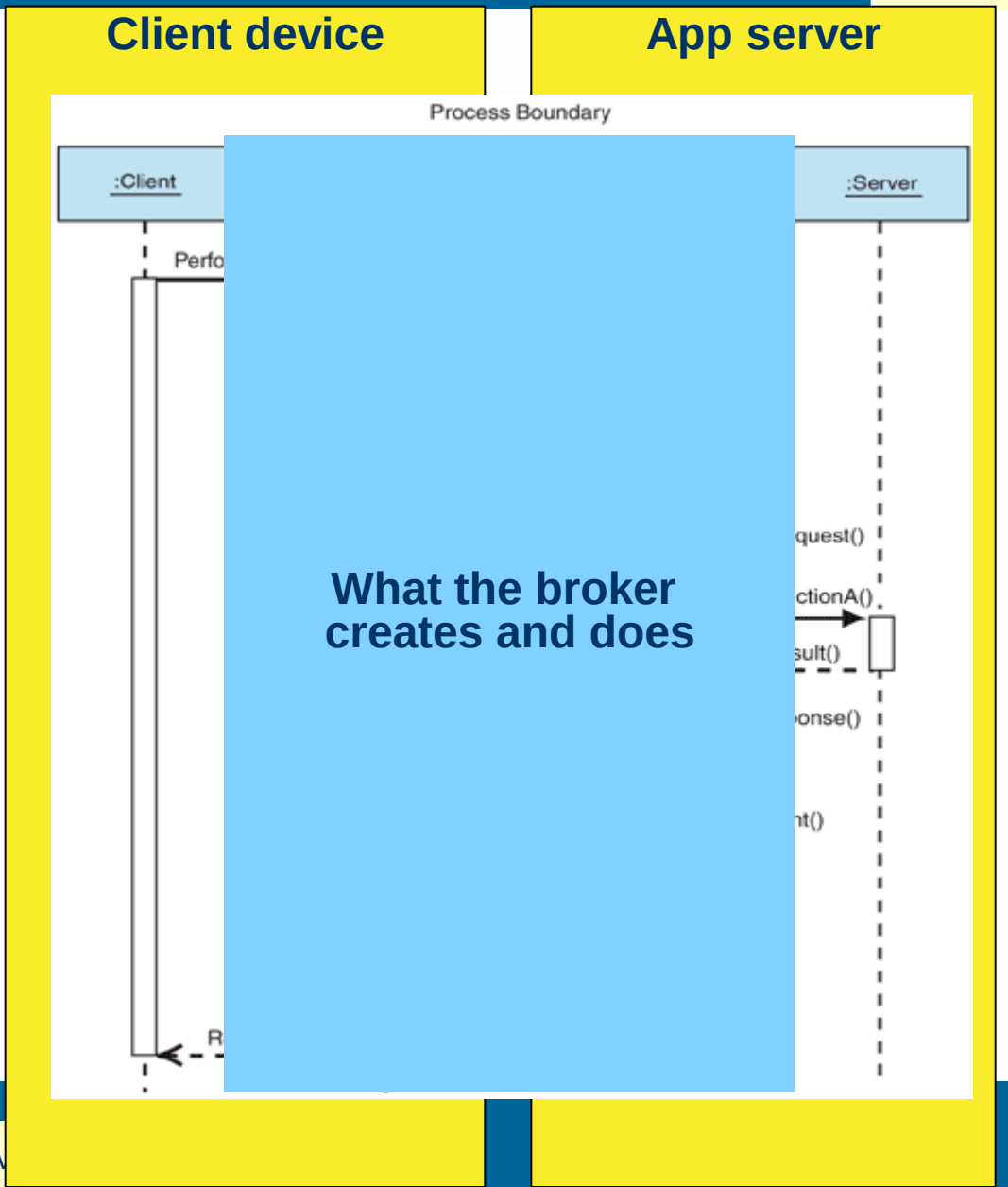
Remember the OO envangelists' ambition

- Turn the world into one big OO program (one name space)
- Intelligent domain objects can run on different machines
- And cooperate as though they run on the same machine.

Object Request Brokers (ORBs)

An Object Request Broker establishes the proxies which manage the remote procedure call

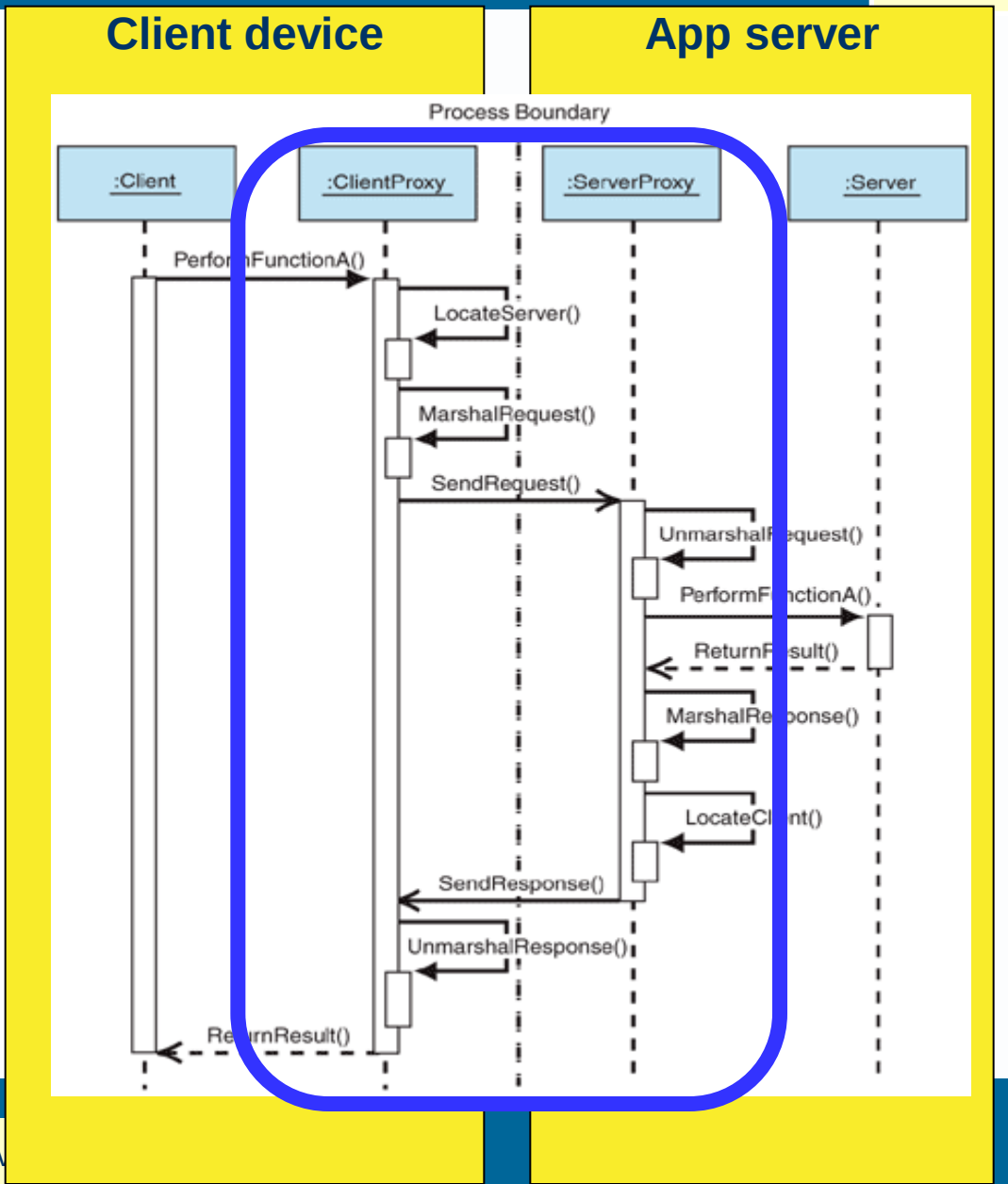
So client and server think they are on the same machine.



Hmm…

Object Request Brokers were supposed to free software designers from having to think about where objects are deployed

Tell anecdote



2002 Fowler's First Law of Distributed Object Design

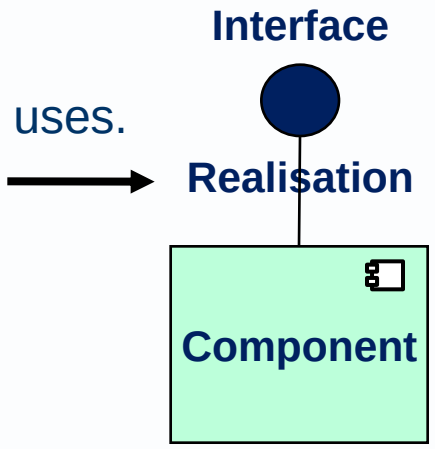
- Don't distribute your objects!
  - Put all your classes in a single "process" [meaning executable]
  - Wrap up fine-grained classes behind Façades
- Unless you are forced to by the physical design
  - By client-server distribution
    - Client device - Web server – App server – Data server
  - Or the need to "scale out" for performance

CBD

- **Component-based *development***
  - Using an ORB to connect distributed objects
  - Architects complained
    - the granularity of distributed objects is too small to be managed
    - inheritance is limited and fragile reuse mechanism, unusable in distributed systems
    - we need help to modularise enterprise applications that maintain large databases.
- **Component-based *design***
  - Designing application components that are much more coarse-grained than distributed objects
  - Nowadays called **micro services**

Programming to an interface

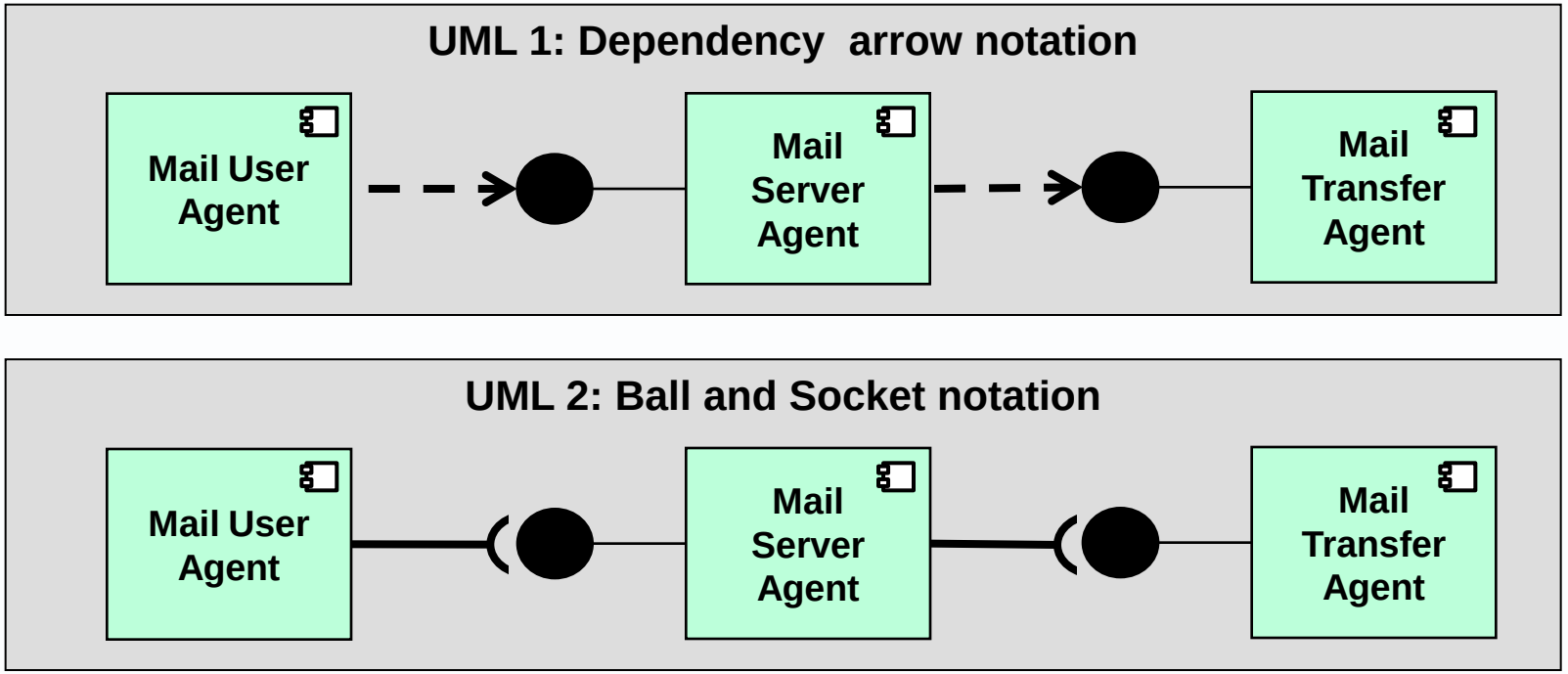
- Method
  - decouple clients from the language a server uses.
  - separate an interface from the work done



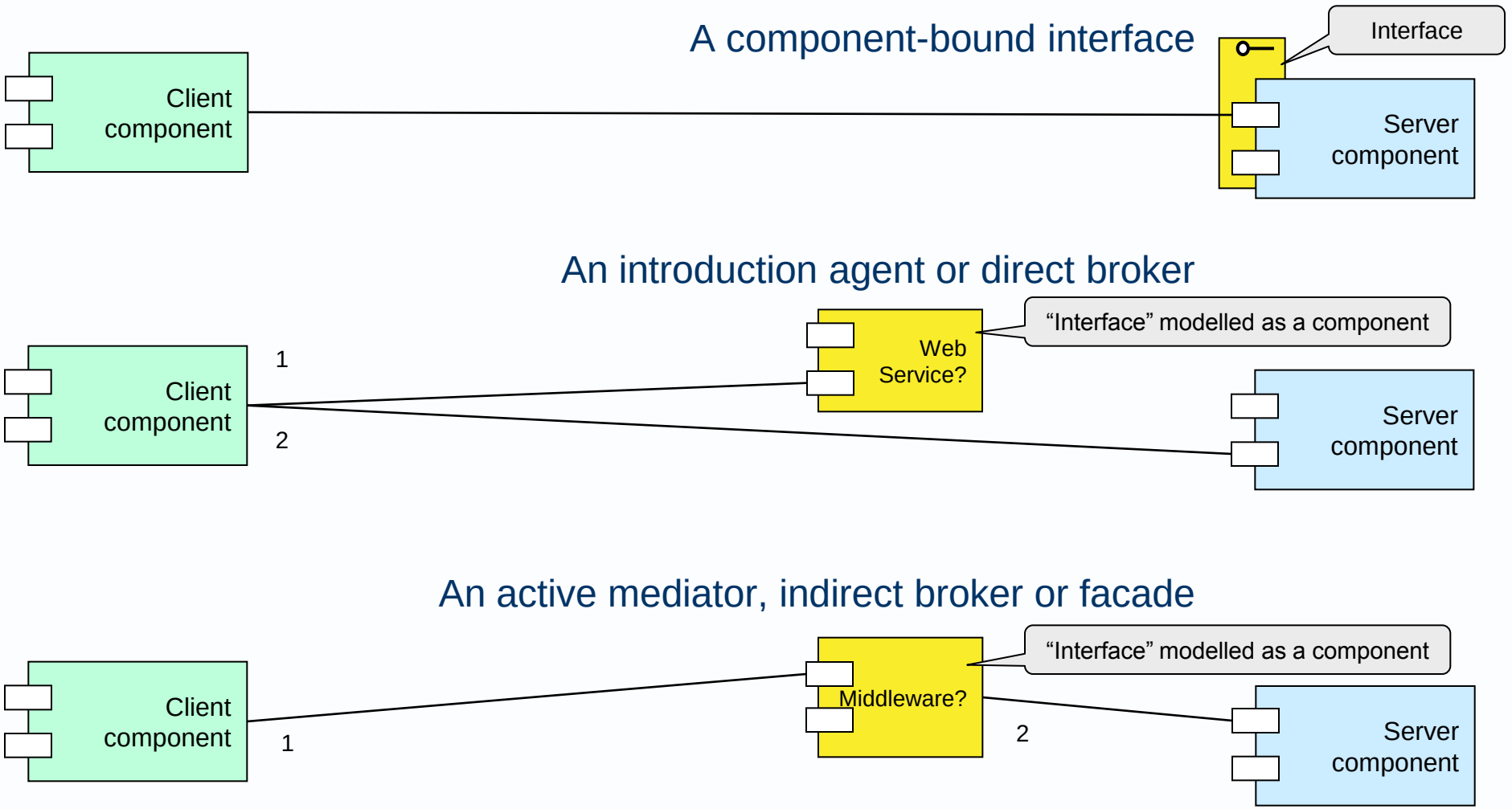
- Tools
  - An IDL of your choice
    - Sun's ONC RPC
    - The Open Group's Distributed Computing Environment
    - IBM's System Object Model
    - Object Management Group's CORBA,
    - WSDL for Web services
  - Latterly WSDL

Modelling interfaces in UML

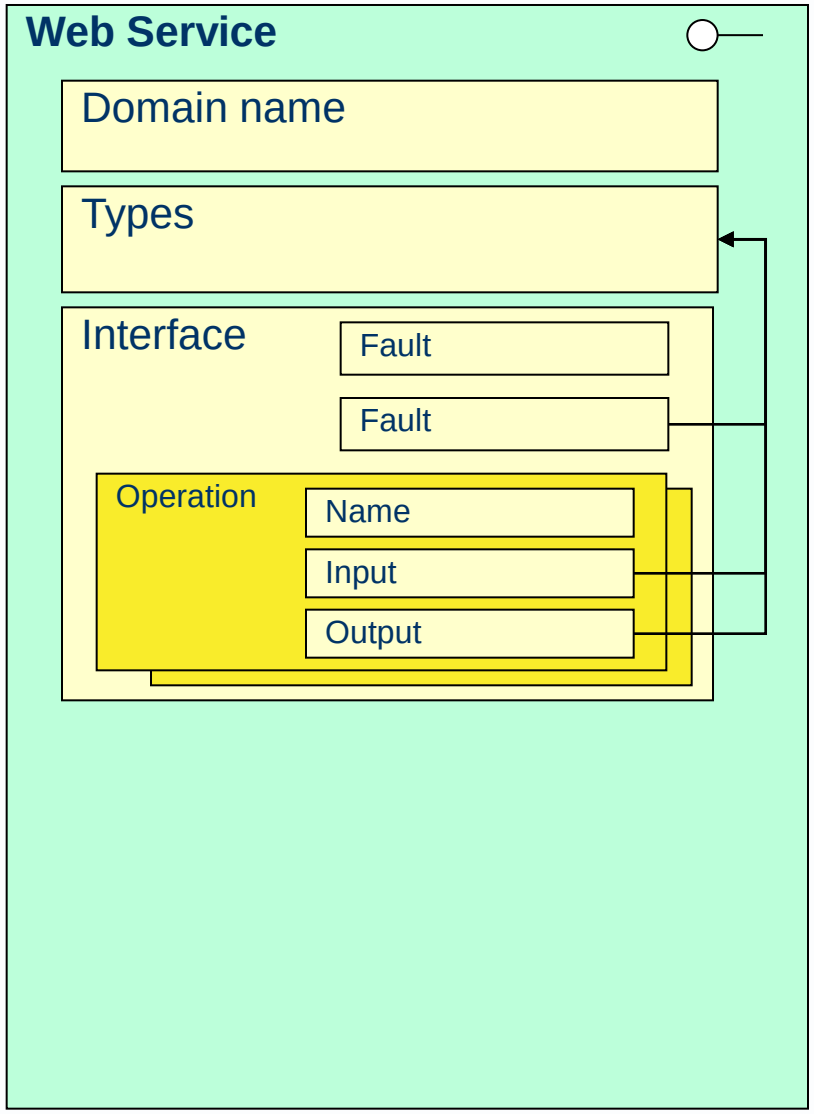
- A component with a required interface desires to meet
- A component with a matching offered interface.



Modelling Interfaces as Components in ArchiMate



Web Services Definition Language (WSDL)



Abstract section

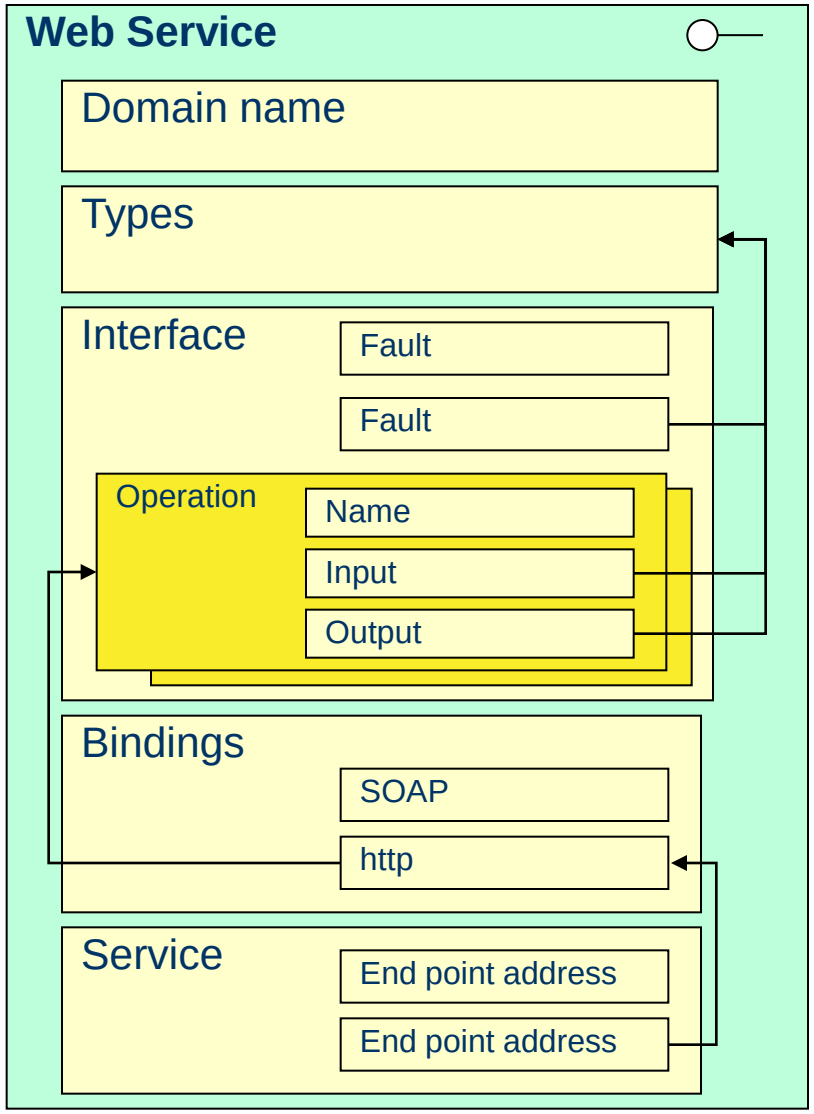
Types: describes the data items used in I/O messages (using an XML Schema)

Interface:

Fault messages

Operations discrete service s/behaviours

Web Services Definition Language (WSDL)



Abstract section

Types: describes data items used in I/O messages (using an XML Schema)

Interface:

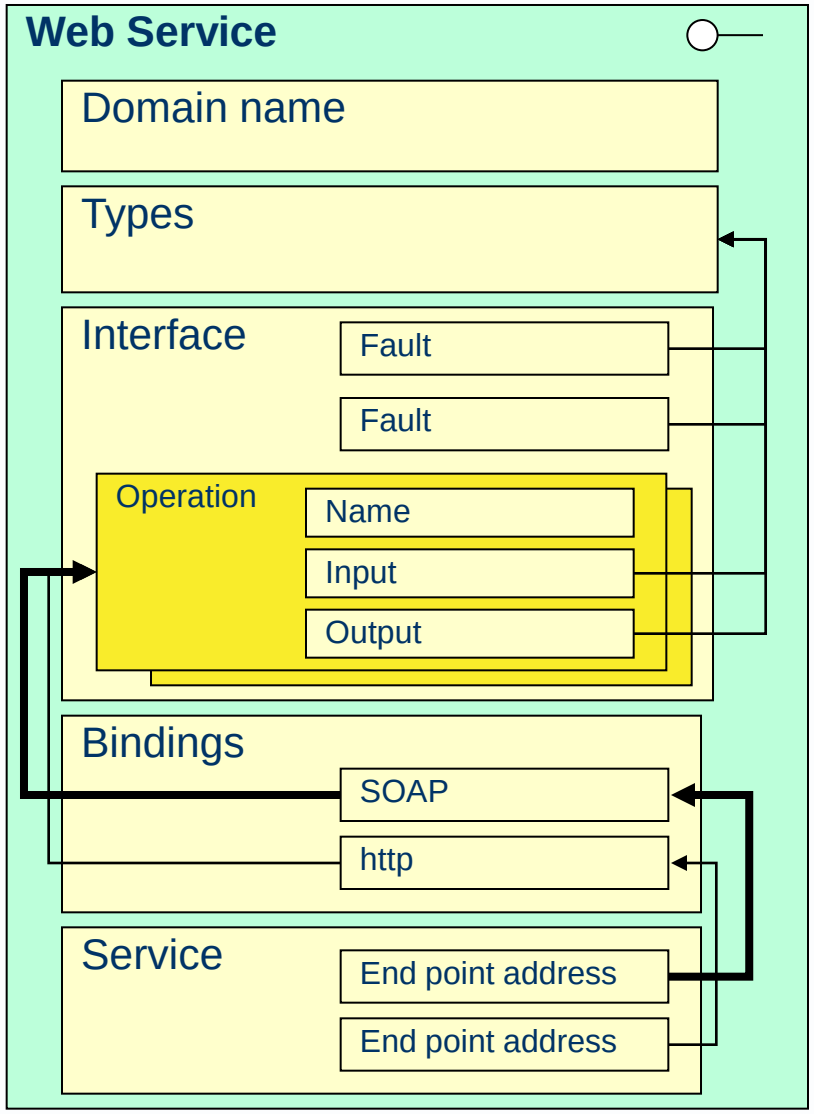
Fault messages

Operations discrete service s/behaviours

Concrete section

Binds operations to the protocols and addresses needed to locate and invoke the operations at run-time.

There can be more than one way to call the same operation



Confusing entanglement of SOA and SOAP c1999

- Microsoft promoted
- **SOA** in reaction against Distributed Objects, Object Request Brokers and the CORBA standard
- WSDL1 in which
  - resources are identified using URIs / domain names
  - web service operations are invoked
  - by sending XML messages
  - via **Simple Object Access Protocol (SOAP)**
  - over HTTP (or perhaps SMTP).
- Thus, Web Services, SOAP and SOA became confused with each other

Separating SOA from SOAP c2003

- Microsoft tended to present SOA as implying that
  - clients should invoke web service operations
  - using the Simple Object Access Protocol (SOAP)
    - A SOAP message is an XML document containing:
      - Envelope - identifies the XML document as a SOAP message.
      - Header - optional
      - Body – contains details of call and response
      - Fault - optional - provides information about errors that may occur while processing the message.

- Many complained SOAP was
  - Not Simple. Not Object-Oriented. Owned by Microsoft!

- So Microsoft gave it to the W3C
  - SOAP 1.2 became a W3C recommendation in 2003
  - And SOAP merely a name.

Changes made in WSDL 2

- Modules in remote systems interact with a web service in the manner prescribed by its description in WSDL

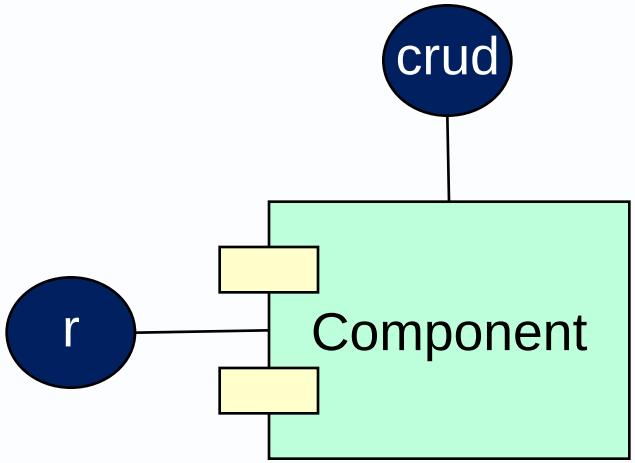
	WSDL1	WSDL2 allows also
Data format	XML messages	JSON
Protocol	SOAP over HTTP (or perhaps SMTP	HTTP directly

- Programmers like to use JSON over HTTP directly

SWAGGER for JSON?
WADL (XML) for REST?

So, the history of software architecture led us to

- Decouple
  - **Components** that do the work
  - **Interfaces** that present services to clients
  - **Services** that can appear in several interfaces



	Service	Interface	Component
Web Services Definition Language	Read Operation	Web Service	Component

What are the qualities of a good service? (After Thomas Erl)

A service conforming to Web Service standards has four qualities: it is an

- abstraction,
- composable,
- **loosely-coupled**
- defined by a contract.

An interface that hides the workings

Useable in a higher level process

By time, location and other ways

Beyond that, a well-designed service should be

- **stateless**
- reusable
- autonomous,
- discoverable

Designers at design time (catalogue)
Client components at run time (directory)!

For simplicity, a service should be transactional as well.

Decoupling part 1

Tightly-coupled
Faster / Simpler

Loosely-coupled
More Flexible

Feature	Early OO design presumptions	Recent SOA design presumptions
Naming	Clients use object identifiers One name space	Clients use domain names Multiple name spaces behind interfaces
Paradigm	Stateful objects/modules Reuse by OO inheritance Intelligent domain objects	Stateless objects/modules Reuse by delegation Intelligent process controllers
Time	Request-reply invocations Blocking servers	Asynchronous messaging Non-blocking servers
Location	Remember remote addresses	Use brokers/directories/facades

COBOL modules
Java objects
CORBA

Web Services

Pick your battles (Craig Larman, “Applying UML and patterns”)

- **“Coupling and cohesion** [after Larry Constantine,1968] are truly fundamental principles in design and should be appreciated as such by all…”

- “It is not high coupling per se that is the problem; it is high coupling to elements that are unstable in some dimension, such as their
  - interface [definition of services provided or required]
  - implementation [vendor-specific technology]
  - mere presence [availability]”

- “If we put effort into “future proofing” or lowering the coupling when we have no realistic motivation, this is not time well spent.”
- “Focus on the points of realistic high instability or evolution.”

Footnotes

- Left overs

Client-server communication using asynchronous servers

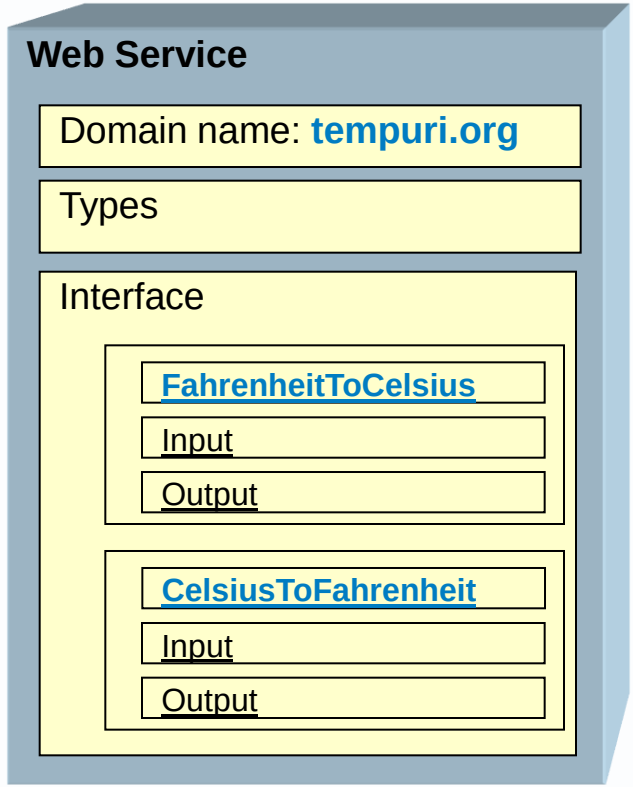
- REQUEST-REPLY using asynchronous message passing
  - Client wants a service,
  - Client sends an invocation message with a unique reference.
  - Client checks its mail box until it finds a reply message with unique reference, or else a timeout

- SUPERVISION OF PARALLEL PROCESSES using asynchronous message passing
  - Client spawns N subordinate processes, sending each a unique reference
  - Client checks its mail box until all process reply or else a timeout.
  - If time out, client kills all subordinates and returns error message to higher supervisor.

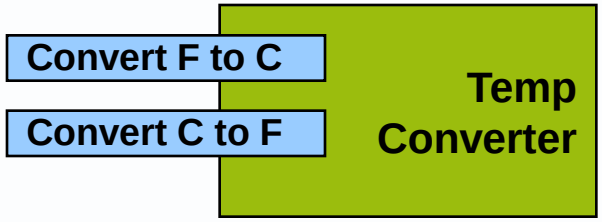
- EVENT SEQUENCE HANDLER
  - Reject out-of-sequence events.
  - Or buffer them until other events mean they fit

After http://www.w3schools.com/webservices/ws_example.asp

- A web service ***interface***
- Published somewhere
- A voluminous WSDL file with a FQN (fully qualified name) for each operation.



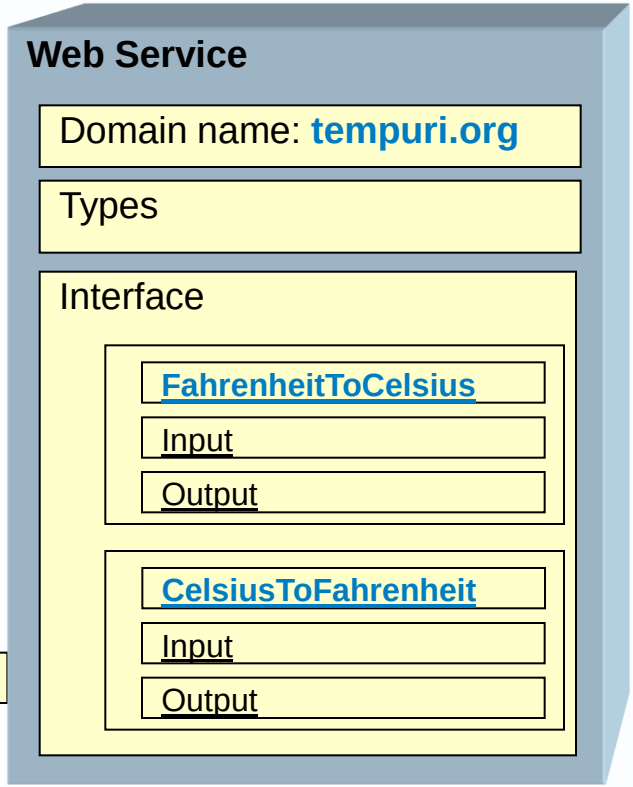
- A web service ***implementation***
- On an app server
  - E.g. A simple VBScript class with two operations
  - A client could invoke an operation at run time
    - w class name, object id. and operation name.
  - But we want to decouple a client from
    - the app server name space
    - the programming language



After http://www.w3schools.com/webservices/ws_example.asp

- At design time, somebody
  - assigns named operations to named domains
  - defines and publishes the logical interface (WSDL file)
  - may import the logical interface into the client
- At run time a client calls an operation using its FQN
  - Protocol
  - Domain name
  - Operation name

```
<soap:operation soapAction="http://tempuri.org/FahrenheitToCelsius" style="document" />
```



- The App Server (Apache, WebSphere, Glassfish) binds this to an operation of the web service implementation