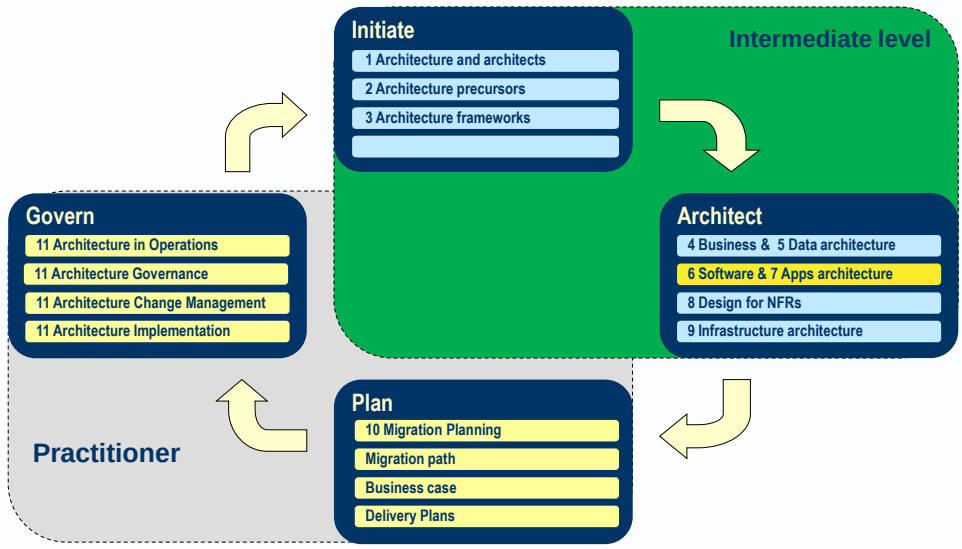
Avancier

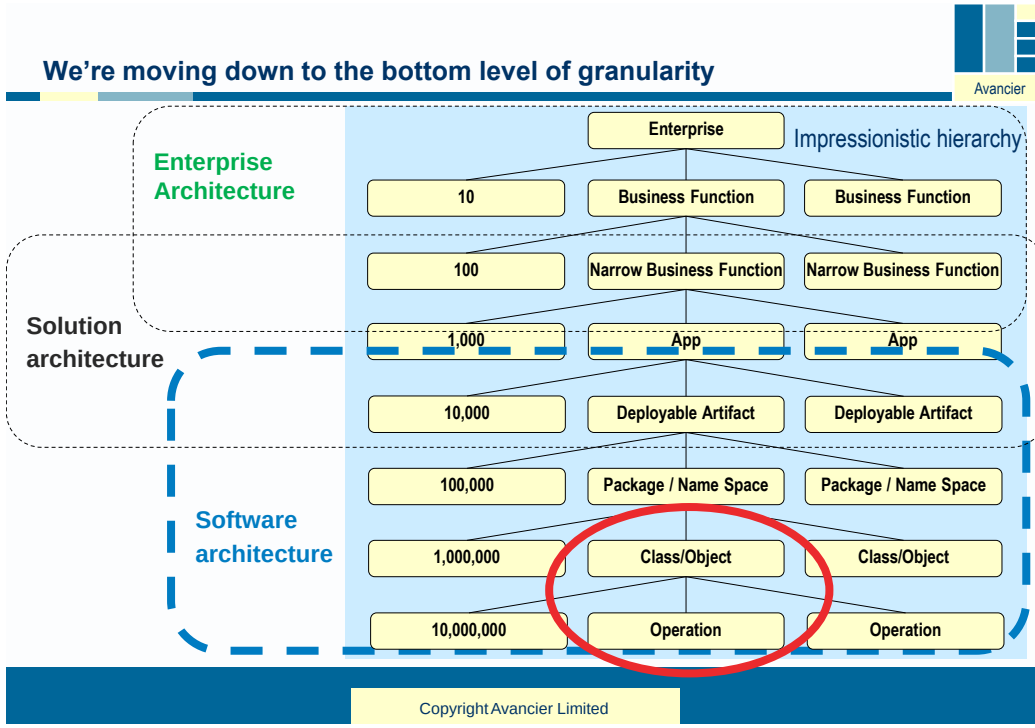
Avancier Reference Model

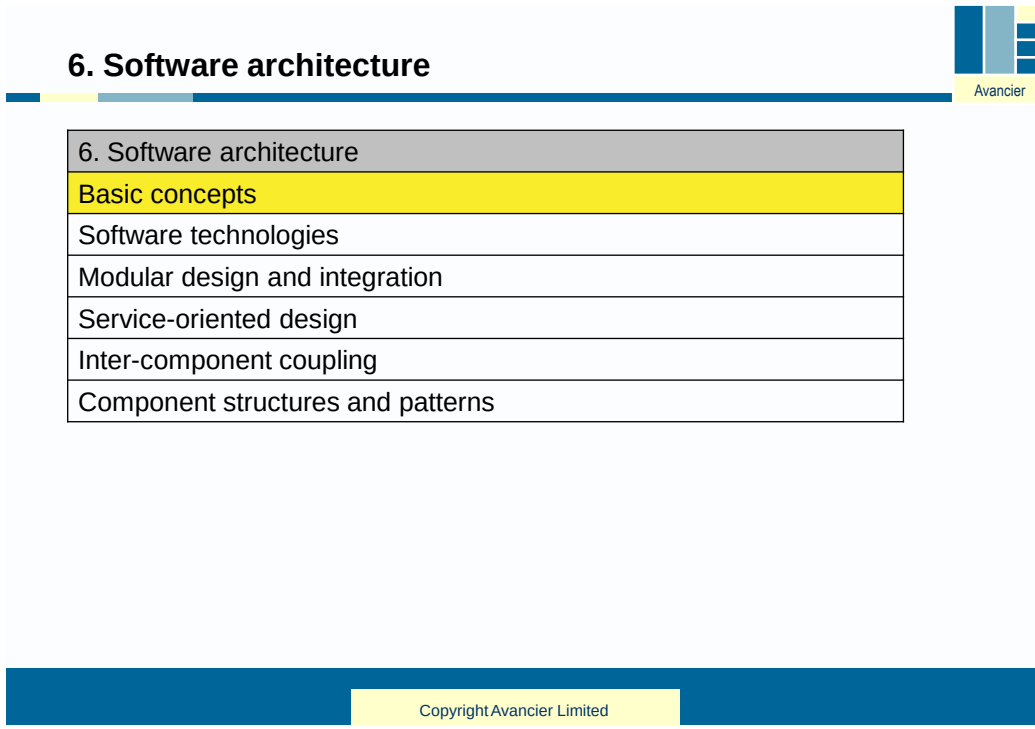
Software Architecture (ESA 6)

6. Software architecture

Avancier

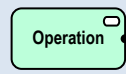
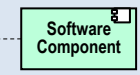



We're moving down to the bottom level of granularity
Avancier
Enterprise Architecture
Solution architecture
Software architecture
Impressionistic hierarchy
Enterprise
10
Business Function
Business Function
100
Narrow Business Function
Narrow Business Function
1,000
App
App
10,000
Deployable Artifact
Deployable Artifact
100,000
Package / Name Space
Package / Name Space
1,000,000
Class/Object
Class/Object
10,000,000
Operation
Operation
Copyright Avancier Limited


6. Software architecture
Avancier
6. Software architecture
Basic concepts
Software technologies
Modular design and integration
Service-oriented design
Inter-component coupling
Component structures and patterns
Copyright Avancier Limited

Conceptual framework

Avancier

	Required behaviour	Interface	Physical Structure
Software			

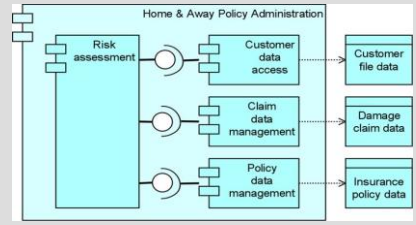
Operation	[a service] that can be requested of a software component. An abstract or virtual operation is an operation for which the signature is defined, but not the internal procedure that implements the operation.
API: Application Programming Interface	[an interface] a collection of automated services accessible by software clients; it identifies services, may provide access to them, and hides what performs.
Software component	[a component] module or object capable of offering automated services. Module is a generic term. Class is the OOP term. An object is an instance of a class; it has identity, behaviour and state; it can perform the general operations of the class and maintain object-specific data item values in its state.

Software Engineering Diagram (TOGAF)

Avancier

- breaks applications into packages, modules, services, and operations from a development perspective.
- enables more detailed impact analysis when planning migration stages, and analyzing opportunities and solutions.
- ideal for application development teams and application management teams when managing complex development environments.

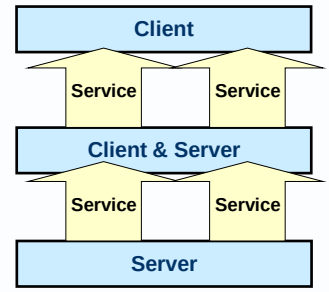
ArchiMate: Application Structure Viewpoint



What do we mean by client and server?

Avancier

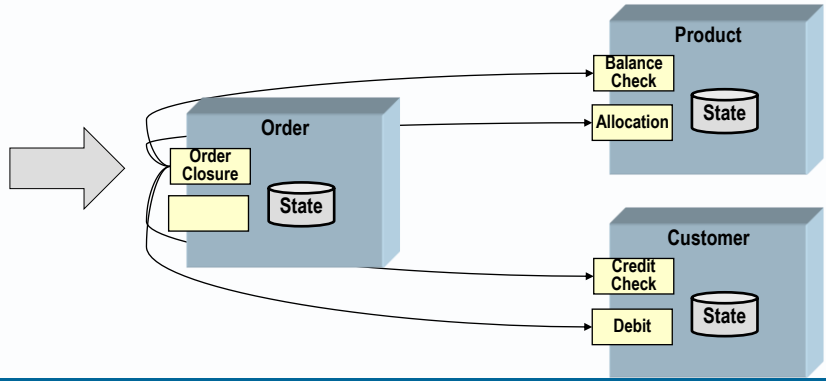
Client component	[a component] that **requests** one or more services from a server component. The request is usually called an invocation
Server component	[a component] that can **provide** one or more services in response to requests from a client. Cluster analysis can be used to group services that are related (e.g. by access to the same data) into one API or server component.



Delegation

Avancier

Delegation	[a process] whereby on component calls on another to do the work, invoking it by passing a message.

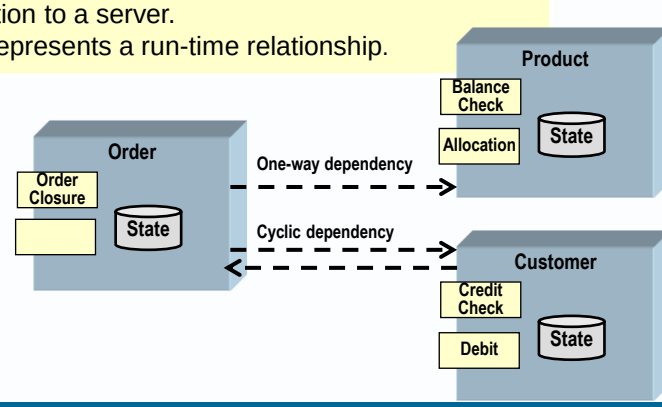


Dependency (an abstraction from the detailed invocations)

Avancier

Dependency

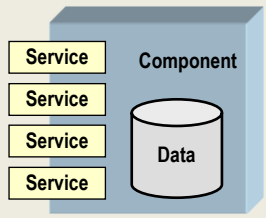
[a mapping] a relationship between two parties in which any change to the depended-on party implies an impact analysis of the dependent party.
Often reflects a client-server invocation, or delegation to a server.
Often represents a run-time relationship.



Stateful component

Avancier

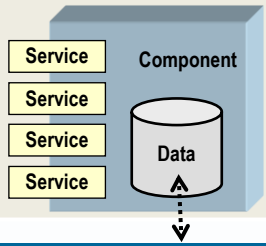
Stateful component



[a component] a component that retains data in its local memory between invocations

The state is lost if the component is removed.

Stateless component

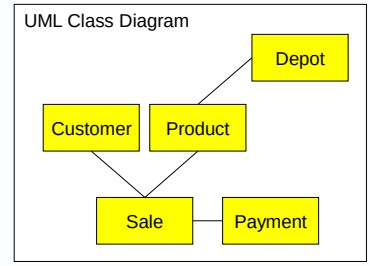


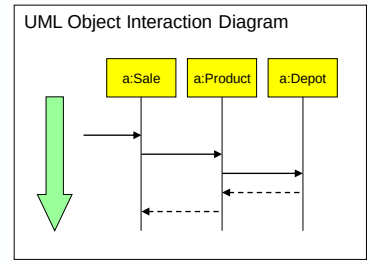
[a component] a component that does not retain data between invocations

Though its transient state can be copied into a persistent data store.

Class diagram

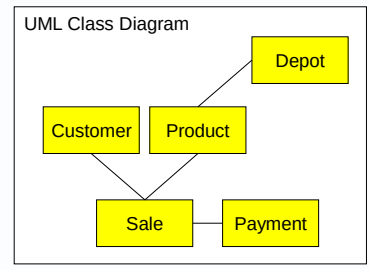
Class diagram	[an artefact] that shows the design-time structure of a software application, showing which components are related and how.
Sequence diagram	[an artefact] that shows how components cooperate at run-time to enable a process. Knowing how a design-time structure will behave at run time is critical to meeting requirements.





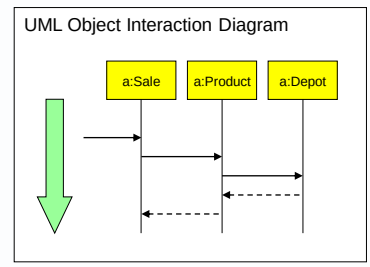
Design time v runtime

- Design time is when an architect describes a structure of components.



- Run time is when those components work and interoperate.

- An architect should have a good idea of how components in a design-time structure will behave at run time, since it is critical to meeting non-functional requirements.



6. Software architecture – end of pass 1

- SHOW RELEVANT MOCK EXAM QUESTIONS

Software technologies

6. Software architecture
Basic concepts
Software technologies
Modular design and integration
Service-oriented design
Inter-component coupling
Component structures and patterns

OOP

Object-Oriented Programming	[a technique] in which components differ from more traditional modules in three ways. ❖A class (a type) specifies objects of the same kind that can be instantiated at run time. ❖An object has a unique *identity* as well as a structure (internal state data) and behaviour (processes or operations). ❖An object (of a given class/type) can inherit behaviour from a super class/type.

Garbage collection

Garbage collection	[a process] that deletes unused objects from memory, which is especially necessary where applications process thousands or millions of objects. (The earliest OO programs handled only a few stateful objects in memory.)

Middleware

Avancier

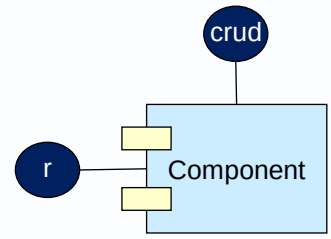
EAI: **Enterprise Application Integration middleware**	[a platform component] designed to ❖store, route and forward messages between application components, ❖transform messages between, protocols and data formats, ❖manage “federated” or distributed transactions and ❖host procedures or workflows that connect distributed components. It can be more complex and slower than point-to-point integration. But advantages where inter-component communication is one to many or many to one, and where the components at either endpoint are volatile.

IDL

Avancier

Interface Description Language (IDL)	A language for defining an API that is, ideally, independent of the technology used to implement the component behind the interface. Enables communication between components in different languages and running on different operating system.

E.g. 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



Web services

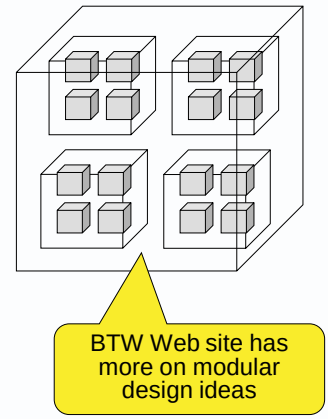
Web Service	[a passive interface] a machine-readable description in Web Service Description Language (WSDL) of one or more web service operations that can be called.
Web service operation	[a service] a discretely invokable action that can be called by components in remote systems as prescribed by a [web service] that defines the operation in terms of ❖parameters expected, ❖data structures returned, and ❖how to access it using web-oriented protocols.

6. Software architecture

6. Software architecture
Basic concepts
Software technologies
Modular design and integration
Service-oriented design
Inter-component coupling
Component structures and patterns

Modular design

Modularity	[a concern] about the ways to divide a “monolithic” system into self-contained, encapsulated components.
Modular design	[a technique] for scoping a component, avoiding duplication between components, separating or distributing components, and integrating components.



Modular integration

Component interoperation style	A paradigm, exemplar or pattern for how components communicate. Four styles, varying from tightly-coupled to loosely-coupled, are **DO, SOA, REST and EDA**.

Tightly-coupled — Interoperation styles — Loosely-coupled

LPC RPC **DO** SOAP **REST** **EDA**
SOA

LPC and RPC

Avancier

Local Procedure Call	[a process] by which one component calls another component running on the same computer. It is simpler, quicker and more secure than a remote procedure call.
Remote Procedure Call	[a process] by which a process on one computer calls a process on another computer. It is more complex, slower and less secure than a local procedure call.

Tightly-coupled — Interoperation styles — Loosely-coupled

LPC RPC **DO** SOAP **REST** **EDA** **SOA**

OOD

Avancier

Object-Oriented Design	[a technique] a kind of modular design and integration that *initially* tended to assume the following features. 1. a component is an object, an instance of a class, identified by an object identifier 2. client/sender and server/receiver objects work in the same name space 3. an object is stateful 4. an object can reuse behaviour by inheriting it from a super type class 5. intelligent domain objects (rather than intelligent process/control objects) 6. client objects communicate with server objects by request-reply invocations 7. a server object can accept only one invocation at once (so blocks others)

DO

DO: **Distributed Objects**	[a technique] a distributed modular design and integration style that employs an object request broker (a kind of middleware). The idea is to scale up object-oriented program design from one computer to many. Software is coded as though all objects are local, and call each other directly, so the distributed system behaves like one object-oriented program.

Tightly-coupled — Interoperation styles — Loosely-coupled

LPC RPC **DO** SOAP **REST** **EDA** **SOA**

CBD

CBD: **Component-Based Development**	[a technique] that can mean either development using the Distributed Objects technique, or designing components that are more coarse-grained than the objects in OO design.

SOA

Avancier

SOA: **Service–Oriented Architecture**	[a technique] that is a more loosely-coupled kind of modular design and integration than Distributed Objects. Typical assumptions are: 1. a component is identified by an internet domain name or URI 2. client/sender and server/receiver components work in different name spaces 3. a component is stateless. 4. a component can only reuse behaviour by delegating to (invoking) another component 5. process/control/workflow components contain some rules 6. client components communicate with server components by message passing 7. a server component can accept more than one invocation at time.

LPC RPC **DO** SOAP **REST** **EDA**

SOA

REST

Avancier

REST: **Representational State Transfer**	[a technique] or modular design and integration style that has all the features listed under SOA and one more. A component can offer only the operations defined in the internet protocol used to invoke it. This further decouples distributed components; it means a client or sender component needs minimal information about the server or receiver component.
WOA: **Web-Oriented Architecture**	[a technique] a label used for a modular design and integration style that tends to favour RESTful and REST-compliant components, wrap up distributed software components behind web services (logical component interfaces) and minimise or avoid use of EAI middleware.

Tightly-coupled Interoperation styles Loosely-coupled

LPC RPC **DO** SOAP **REST** **EDA**

SOA

EDA

Avancier

EDA: **Event-Driven Architecture**	[a technique] that decouples the client/senders of news or update events from servers/receivers. Any component can receive or read any event/message published or posted by any other component. It uses either the publish and subscribe functions of a middleware technology, or else a shared data space. (Aside: an EDA may use a Complex Event Processing engine that monitors combinations of events, and triggers other events in predictable way.)

Tightly-coupled — Interoperation styles — Loosely-coupled

LPC RPC **DO** SOAP **REST** **EDA**

SOA

6. Software architecture

Avancier

6. Software architecture
Basic concepts
Software technologies
Modular design and integration
Service-oriented design
Inter-component coupling
Component structures and patterns

A module offers a service or several related services

Avancier

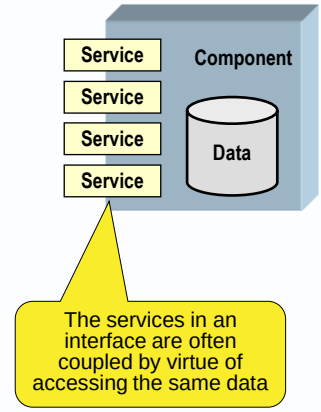
Cluster or affinity analysis

Techniques for finding and grouping items based on characteristics they share.

An aim is to encapsulate cohesive items in one component, and minimise the couplings between components.

Cohesive means closely related, inter-dependent or tightly-coupled by time, location, acquaintance, protocol or other ways.

[See tight-coupling for more detail.]



SOD

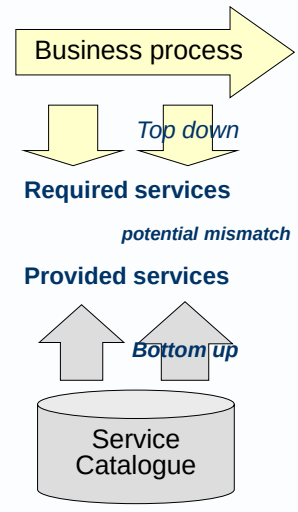
Avancier

Service-oriented design

[a technique] that matches required services to available services.

Required services are discovered though decomposition of high-level business process and use cases.

Available services are discovered in some kind of services catalogue and invokable across a network via some kind of services directory.



Aggregation of services

Service catalogue	[an artefact] a list of services designed to help designers find and reuse them, in which each service is defined by a service contract. It may list interfaces or components that cluster several services and/or discrete services.
Service directory	[an artefact] a service catalogue that a service consumer can use at run time to find how and where each service can be accessed.

SOD (according to Thomas Erl)

Service quality

A characteristic of a service in a well-designed architecture.

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

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

An interface that hides the workings

Useable in a higher level process

Beyond that, a well-designed service should be

- stateless
- reusable,
- autonomous,
- discoverable

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

These eight qualities were suggested by Thomas Erl; and for simplicity, a service should be transactional as well.

SOD challenges

Service-oriented design challenge	[a planning concern] about service-oriented design: Notably: •service ownership, •maintenance of shared services, •versioning strategy. •governance of the service catalogue, •avoiding vendor lock in, •management of service-oriented design methodology across the enterprise. As a service consumer, you don't want service owner to change the interface unless you ask, or refuse to change the interface when you do ask.

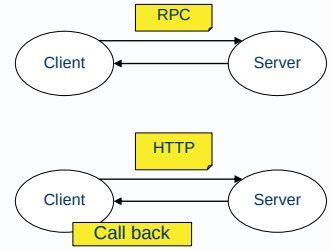
6. Software architecture

6. Software architecture
Basic concepts
Software technologies
Modular design and integration
Service-oriented design
Inter-component coupling
Component structures and patterns

Coupling by time

Avancier

Synchronicity	How work, divided between two components, is scheduled. The two components are called client and server below, but could be called sender and receiver, or prior and successor.



Coupling by time

Avancier

Synchronous	1: A **request-reply** style; a client must wait for a server to reply before continuing. 2: A **blocking style**; a server serves one client at a time, turning away any other client who attempts to request a service. The caller and responder hold a channel open, blocking others from using it. Aside: 1: is the usual invocation from one COBOL module to another or one Java object to another. 2: is the usual invocation style in CORBA.
Asynchronous	1: A **fire-and-forget** style in which a client does not wait for a server to reply, so can carry on to do other things. A call-back mechanism may be needed. 2: A **non-blocking** style in which a server can accept requests from several clients before responding to the first. Typically, the responder has a queue of incoming messages and releases the channel after a message is received. Aside: 1. is the usual style in email conversations. 2. is the usual style of Web Services.

Summary

Often faster and/or simpler

Often more flexible, but more complex

Avancier

Feature	Tightly coupled	Loosely coupled
Time	Parties communicate synchronously and are deployed together.	Parties communicate asynchronously and need not be deployed together.
Location	Parties know each others' locations.	Parties do not know each others' locations
Data types	Parties exchange complex or strong data types.	Parties exchange only simple or weak data types.
Version	Client components are disabled when a new server component version is released	Client components are not affected when a new server component version is released
Protocol	A client must use one protocol to call a service.	A client can use any of several protocols to call a service.
Transaction management	A process involving two parties will ensure consistency	A process involving two parties will be divided at the expense of consistency

Decoupling techniques

Often faster and/or simpler

Often more flexible, but more complex

Avancier

Feature	Tight coupling	Decoupling techniques
Time	Request-reply invocations Blocking servers	Asynchronous messaging
Location	Remember remote addresses	Use directories
Naming	One name space Shared object identifiers	Separate name spaces Programming to interfaces, REST
Paradigm	OO inheritance Intelligent domain objects Stateful objects	Delegation Control objects Stateless objects
Data types	Complex data types	Simple data types
Version	Version dependency	Design to avoid version dependence Apply the open-closed principle
Protocol	Protocol dependency	Design for multiple protocols
Integrity constraints	ACID transactions	Compensating transactions and eventual consistency (BASE)

6. Software architecture

6. Software architecture
Basic concepts
Software technologies
Modular design and integration
Service-oriented design
Inter-component coupling
Component structures and patterns

Sources include

- Network-based architectural styles (after Roy Fielding)
- OO design patterns (after the Gang of Four)

Component structure or pattern

Pattern	[a structure] model or form that can be reused in similar situations to address similar issues, e.g. to organise components so as to cooperate to complete a higher level process.

Design pattern

Avancier

Software design pattern	[a pattern]: A common structure for modular design and/or integration. The use of design patterns encourages consistency and reduces the risk of reliance on an individual designer. Design patterns often come in pairs; the choice between obverse and reverse patterns is made by trading off their pros and cons.

Façade – an interface design pattern

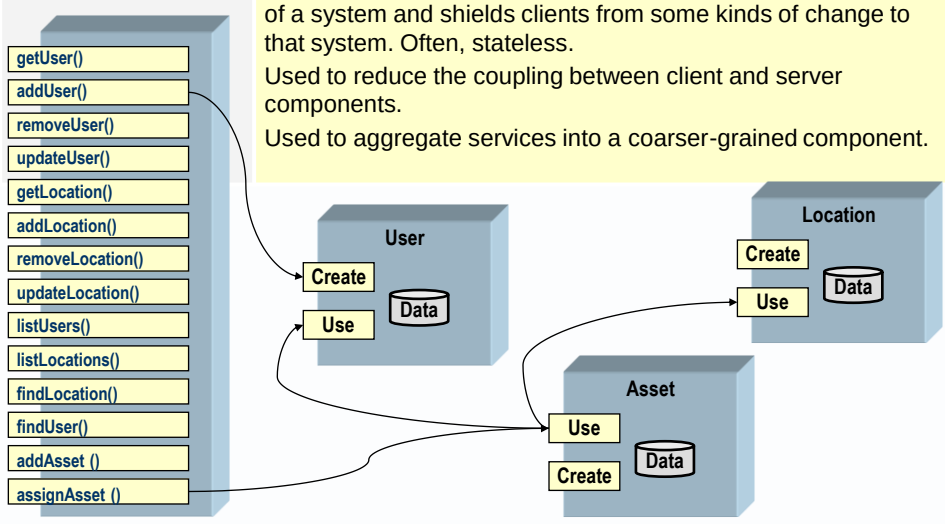
Avancier

Façade

A frontage to a building. An interface component that sits in front of a system and shields clients from some kinds of change to that system. Often, stateless.

Used to reduce the coupling between client and server components.

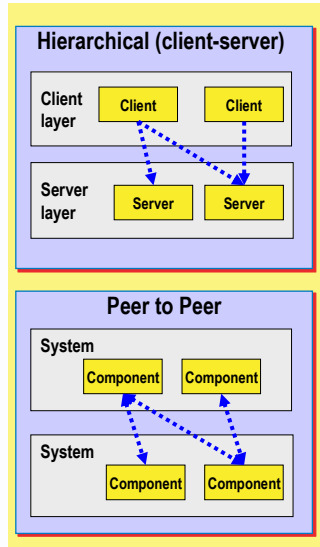
Used to aggregate services into a coarser-grained component.



Design pattern: Hierarchy v Peer to Peer

Avancier

Hierarchical structure	[a pattern] or directed graph, in which a client of component X cannot also be a server of component X.
Peer-to-peer structure	[a pattern] with no hierarchy, meaning that any party can be both a client and server of any other.

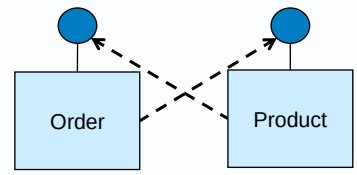


Cyclic dependency

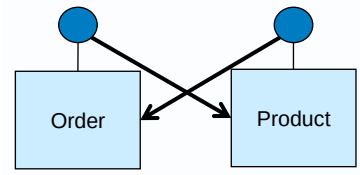
Avancier

Cyclic dependency	[a pattern] in which two or more components depend on each other. Considered to be an architectural flaw - fragile and unstable - difficult to understand and maintain. Software with many cyclic dependencies is said to undermine testability, parallel development, and reuse. However, coarse-grained components are often co-dependent.

UML "dependency" arrows (often reflect uses)



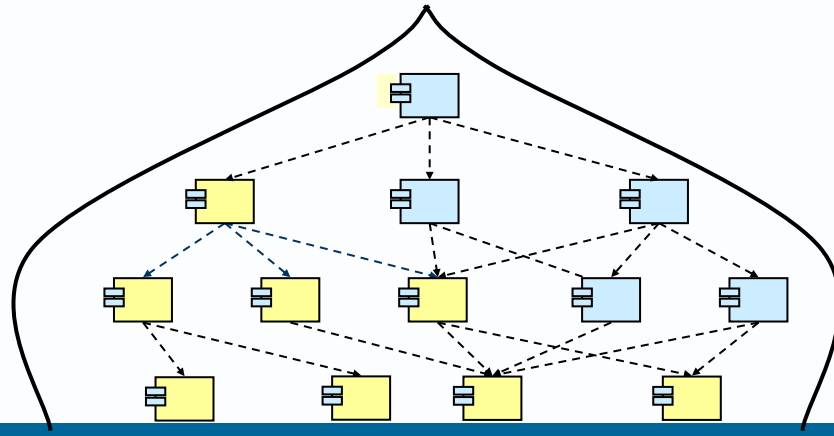
The reverse of ArchiMate "used by" arrows!



Hierarchical (non-cyclic) dependency

Avancier

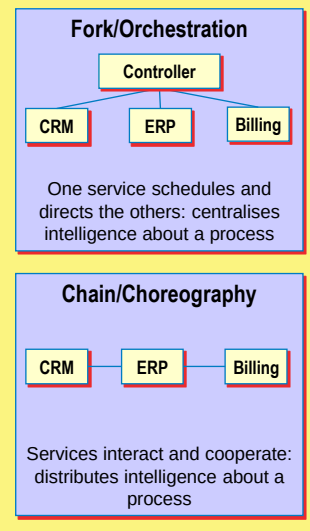
Hierarchical (non-cyclic) dependency	[a pattern] with no cyclical dependency; a relationship in which higher-level components depend on lower-level ones, but not vice-versa



Design pattern: Fork v Chain

Avancier

Orchestration (aka Fork) pattern	[a pattern] that centralises intelligence about a process sequence in a workflow controller that supervises and orchestrates the procedure. It manages the sequence of activities by invoking components in turn.
Choreography (aka Chain) pattern	[a pattern] that distributes intelligence about a process sequence between several entity or domain components. Each component does part of the work, then calls the next component (resembles a pattern called pipe and filter.)

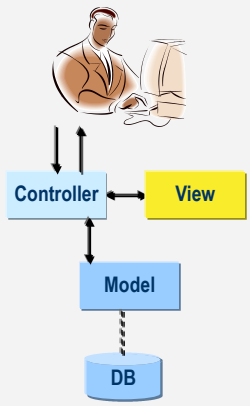


6. Software architecture – end of pass 2

- SHOW RELEVANT MOCK EXAM QUESTIONS

An early "OO" Design pattern: MVC

Model-View Controller (MVC)

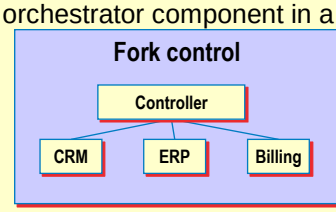
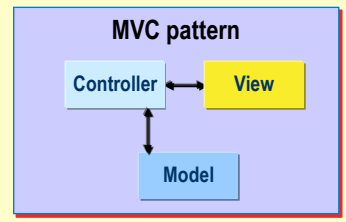


A pattern for software modularity that separates
- user interface "views" from
- the state of the data entities that "model" the world monitored by the system.

There are many variations on how the M, V and C modules interoperate.

Two places a controller appears

Avancier

Controller	1. The manager or orchestrator component in a Fork pattern  2. A component in the MVC pattern, the intermediary between the user interface and the data. 

OO design patterns

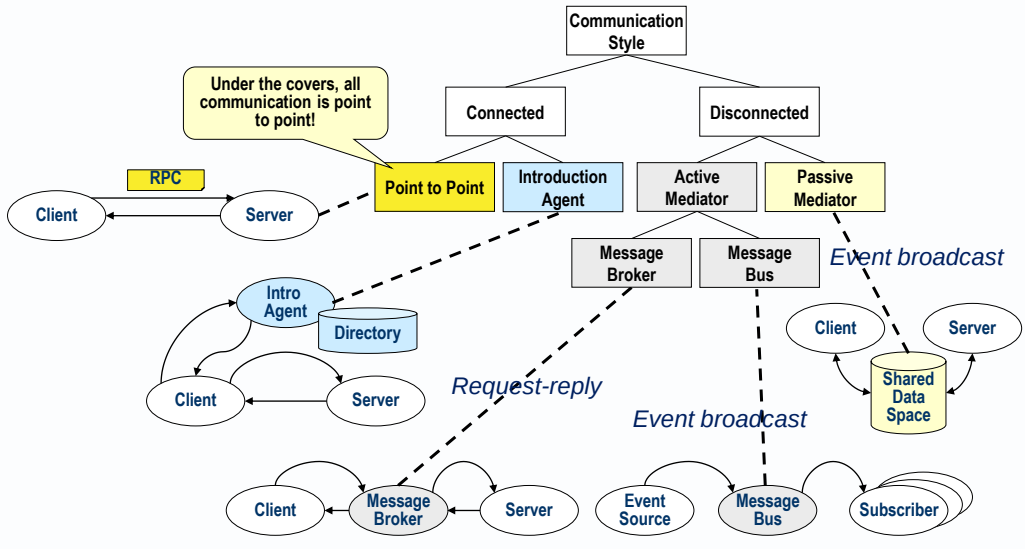
Avancier

OO Design pattern	A pattern used in the design of object-oriented (OO) software. Usually solves a common design problem by placing a broker component between client and server components. A pattern for the modularisation of OO classes that is usually presented as a class diagram. This may be supplemented with an object diagram or interaction diagram. The most famous reference is “Design Patterns: Elements of Reusable Object-Oriented Software” [Gamma et al. 1995]. Some of their patterns are generalisable and useful outside of OO software, including Façade and the patterns below.

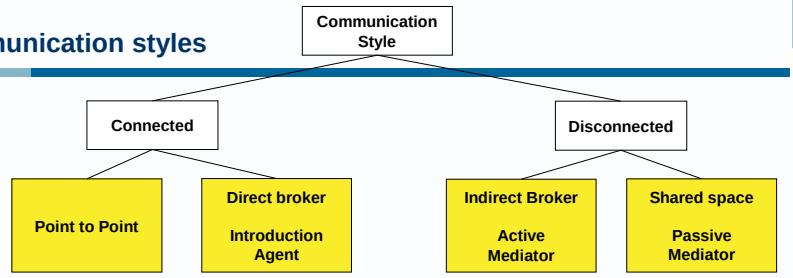
After façade – other common OO design patterns include

Singleton	A component (or class) with only one instance (or object).
Proxy	A surrogate for a distributed component. Used in distribution of code between different name spaces. Simplifies the interoperation of distributed components in DO and SOA interoperation styles, and in CORBA, DCOM and Web Services technologies.
Adapter	A wrapper that converts a provided service into a required service. Facilitates the reuse of existing technologies.
Observer	A component that monitors the state of another component. Curiously, the subject of observation may be an object.

Communication styles – summary overview

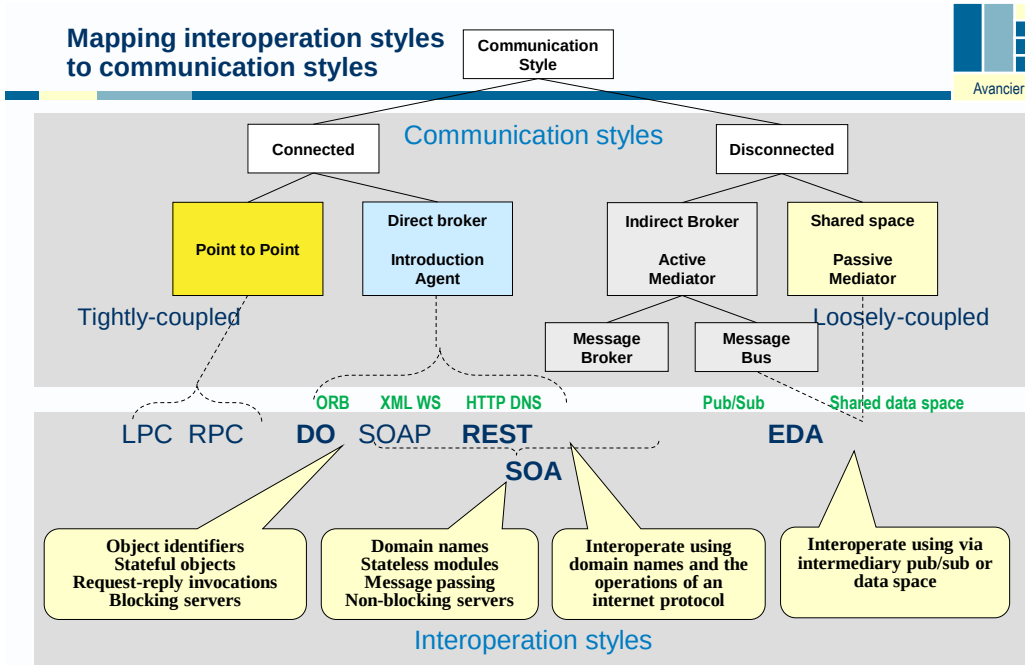


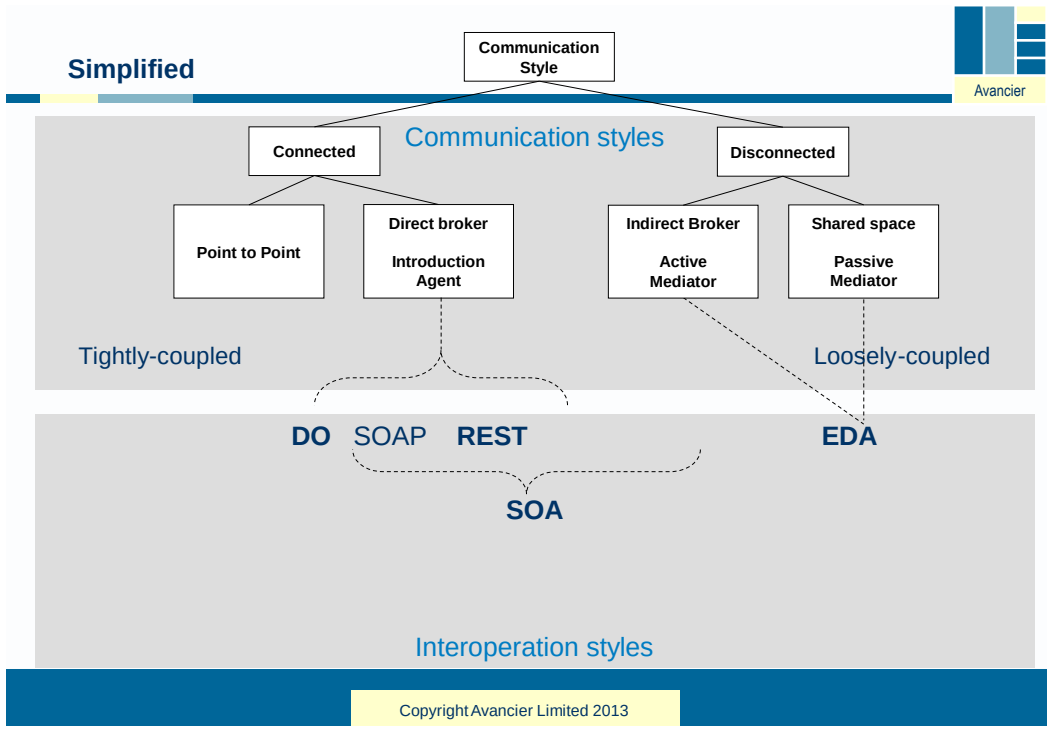
4 communication styles

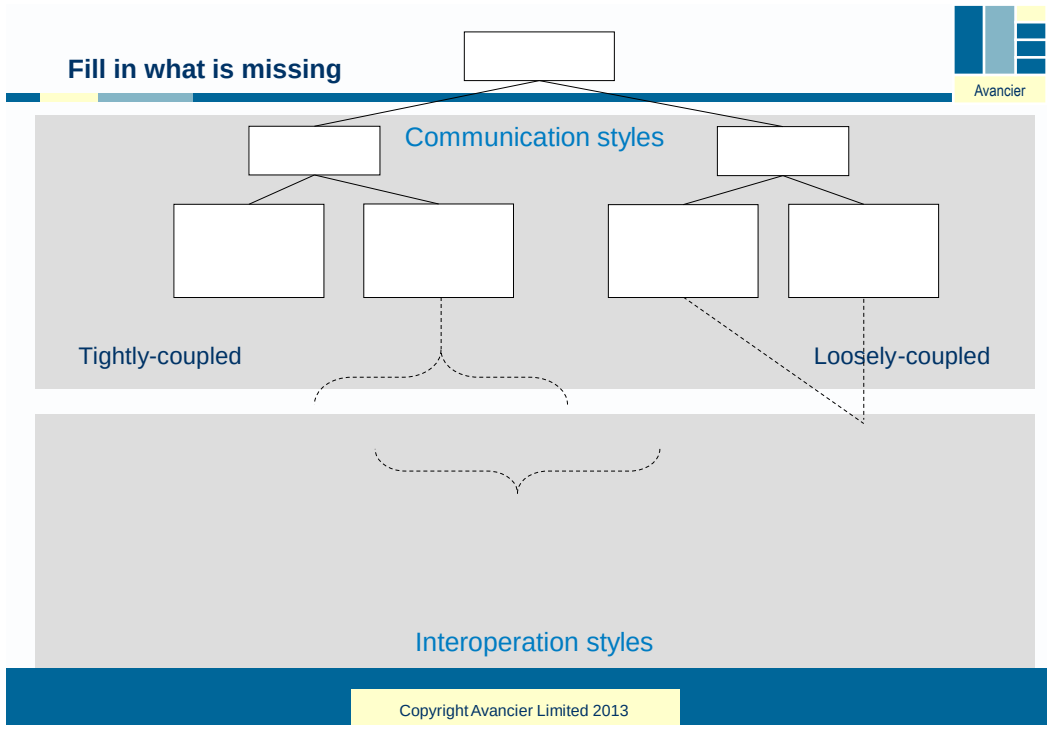


Component communication style	The manner in which components interact (as client and server, or sender and receiver), which can be direct or via intermediaries. There are three broad categories: •Point to point •Introduction agent •Mediator (active or passive)

Mapping interoperation styles to communication styles




Simplified
Communication Style
Avancier
Connected
Communication styles
Disconnected
Point to Point
Direct broker
Introduction Agent
Indirect Broker
Active Mediator
Shared space
Passive Mediator
Tightly-coupled
Loosely-coupled
DO
SOAP
REST
EDA
SOA
Interoperation styles
Copyright Avancier Limited 2013


Fill in what is missing
Avancier
Communication styles
Tightly-coupled
Loosely-coupled
Interoperation styles
Copyright Avancier Limited 2013