Avancier

Avancier Methods (AM)

Guidance and Techniques

Modularity in activity systems
(the restaurant analogy)

Challenges in the design of activity systems

Avancier

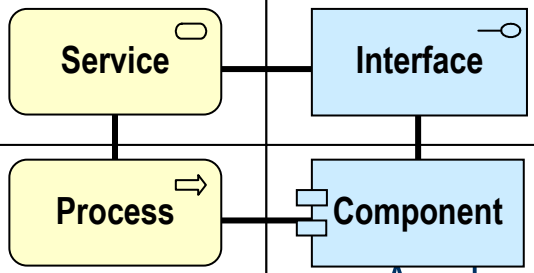
- Not only how to design and build activity systems, both
  - human activity systems (businesses) and
  - computer activity systems (applications).
- In terms of
  - Business Components
  - Application Software Components
  - Technology Platform Components
- But also
  - how to wade through the swamp of the words we use
    - (like “component”)
  - to find whether we are talking about the same thing.

Modularity in activity systems

Avancier

Something customers want from the system, or the system wants from suppliers.
A result of processes, but defined in a service contract without reference to the internal logic of processes used.

A collection of services (to the left) that is required or provided by one or more components (below).

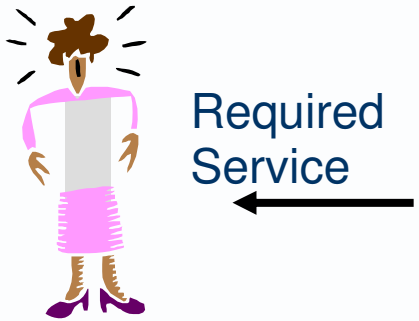


What happens.
A logical sequence of activities that ends up delivering a service at some level of granularity.
Executed by components.

A subsystem that does work, executes activities.
A cohesive group of related but distinctly invokable activities, encapsulated behind an interface.

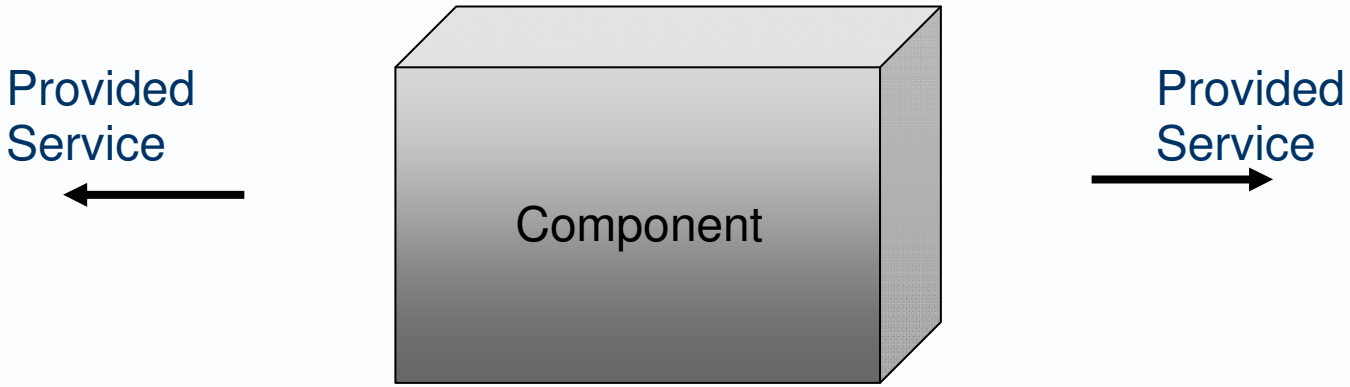
Client components require services

- You need a meal
- A component playing a **client** role requires a **service.**



Server components provide services

- Restaurants provide meals.
- Various **components** (playing a **server** role) can provide a service.



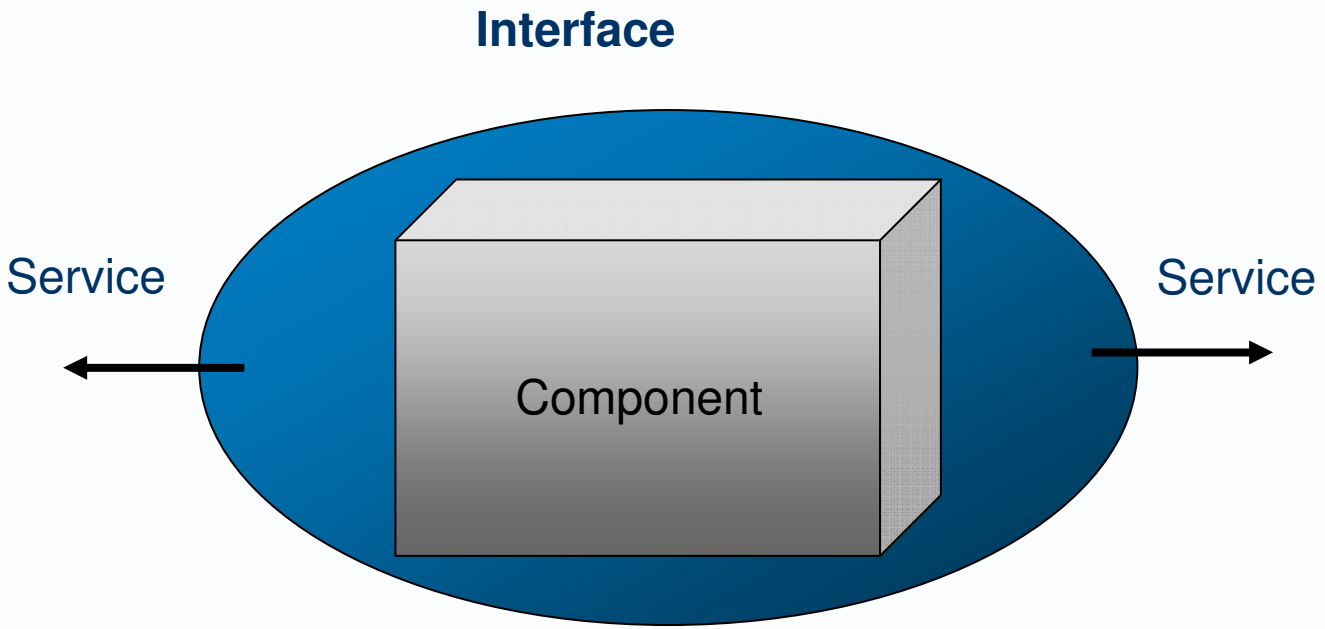
Service contracts are published in interfaces to components

Avancier

- Restaurants publish a service catalogue

Starters
Mains
Desserts…

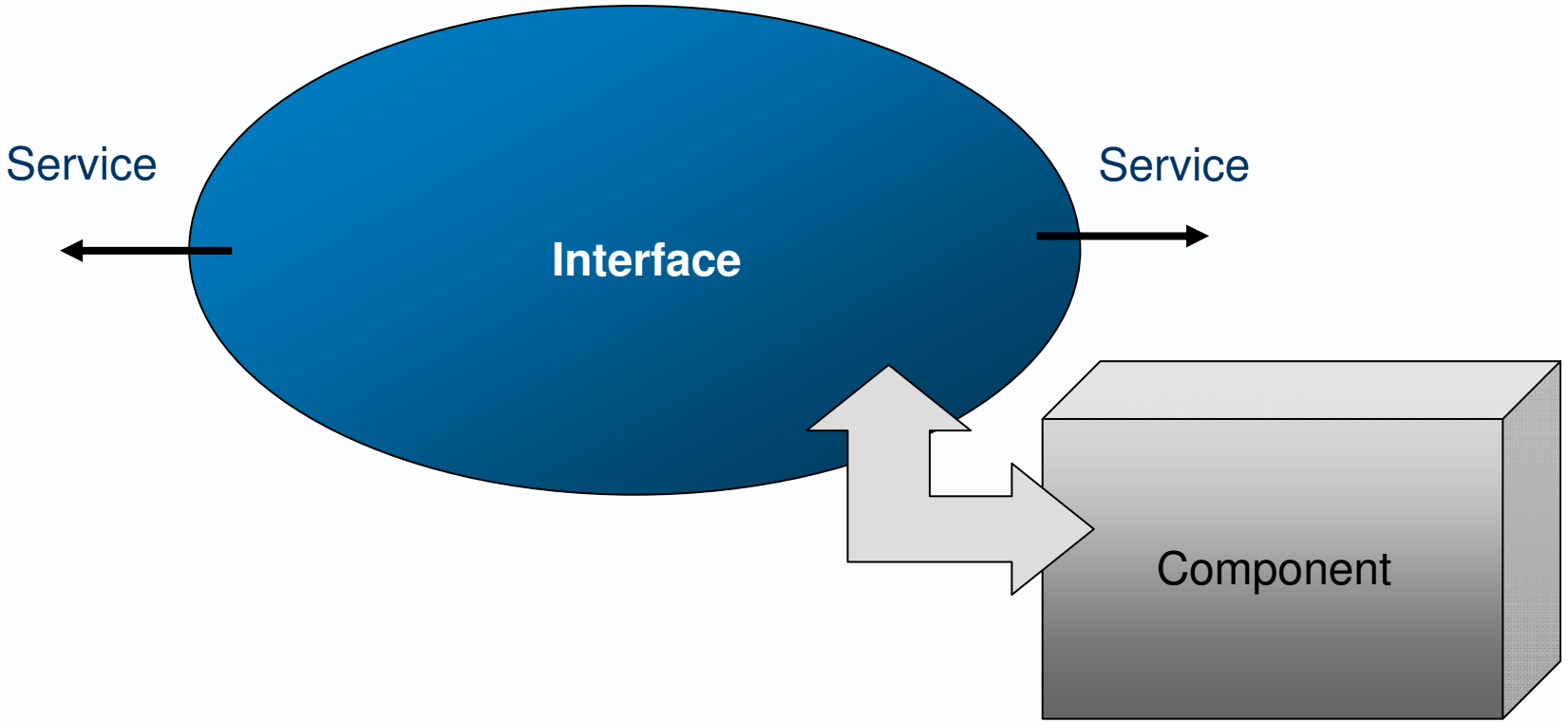
- A list of services can be published as an **interface** to a component



Service contracts may be separated (in catalogues and directories) from service delivery (in components)

Avancier

- Restaurants can publish their menus away from the restaurant location
- A list of services can be separated from the component(s) that execute the services.



In our restaurant

- Services
  - menu items
- Interfaces
  - menus
- Processes
  - procedures the waiter and kitchen staff follow.
- Components
  - waiter, chef and oven.

Governance of services

Avancier

- Governance of their menus is the restaurant's responsibility.
- Customer's expressed requirements have a strong influence, but they are not the only criteria used to decide which services are listed and managed, and how they are specified.
- So governance of services needs
  - Artifacts
    - The service catalogue and interface provided.
  - Organisation
    - Roles needed to create, modify and delete services in the Interfaces, and authorise those changes.
  - Processes
    - Procedures for change requests, impact analysis, approvals etc.

Strictly – our illustration is too simplistic

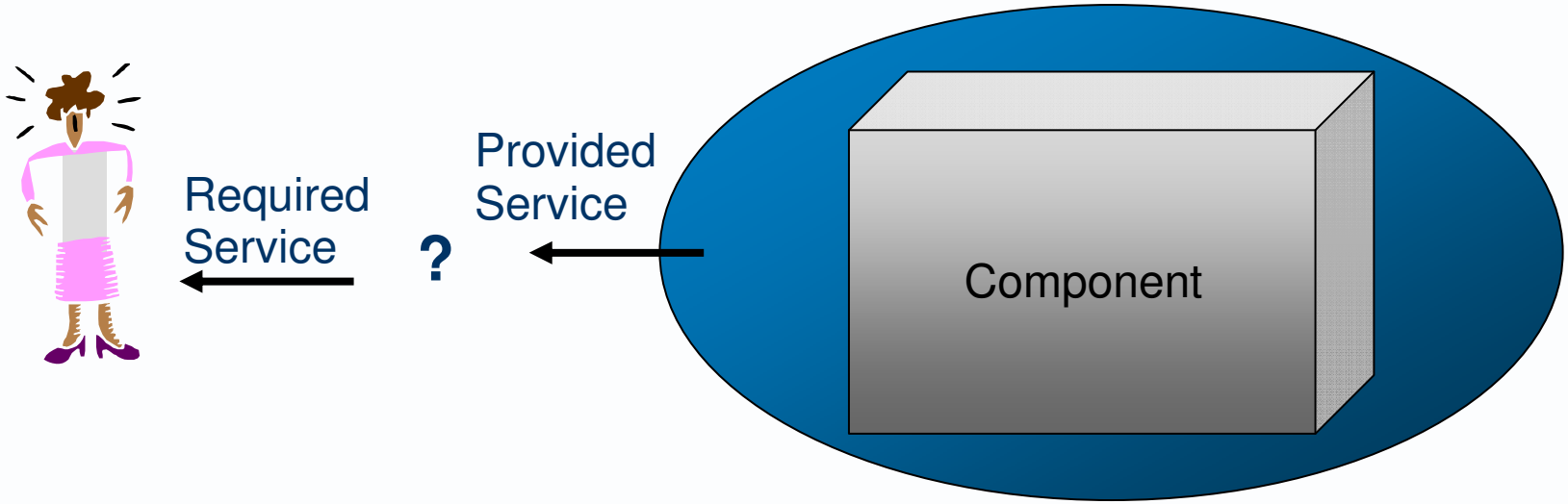
Avancier

- We have had a lot of trouble with the word interface
- In the end, we decided
  - The restaurant menu is an interface
  - The waiter acts as an facade to the back end components
- And we can distinguish:
  - **Catalogue:** A list of services that is managed/governed.
  - **Directory:** A list of services with their addresses and how to find them
  - **Interface:** A list of services required or provided by a system or component
  - **Façade:** An indirect broker or mediator component, *though which* services can be invoked by a client

Required service must match provided service

Avancier

- You accept a menu item description and the price
- A client must match their required service to the **contract** of a provided service.



Every component needs a work place

- A Restaurant needs a work place with the necessary resources.
- A component needs a **host computer** with memory and processor that are sufficient.

A network is needed to reach the work places

- Roads are needed to reach the restaurant.
- A **network** is needed to reach a computer that hosts a component.

ASIDE

- To be shared, a service must be locatable by remote clients
- To be locatable, the server component must sit on a server node of a computer network.
- The server component must have its own local memory on the server node
- The network must be a wide as the enterprise within which clients request the service.
- This doesn't imply any specific network technologies or protocols. In SOA however, most assume the server component can be accessed via Internet Protocols.

A work place may contain several instances (actors) of component type (role)

- From the restaurant's team of waiters, one must be selected to serve a menu item (or perhaps a whole meal)

- The computer may host
  - a **singleton** server component
  - several uniquely-named server components
    - (akin to objects of a class in OO)

A server component provides a service via a channel at the work place

- A waiter sits you a table in the restaurant to offer his services
- A server component provides a service to its clients through **a port** on a networked computer.

ASIDE

- A client needs various facts to find a service in a name space in the memory of a server on a network.
- These facts vary, depending on the degree of coupling by location and coupling by programming style.
- Put aside for now
  - How the client locates (directly or indirectly) the server component.
  - Whether the client needs a physical address or logical address, a single address or a hierarchy of addresses.
  - Whether a client uses a broker to communicate with the server.

Clients and servers must share language and protocols

- You must talk to the waiter in a language he understands.
- A client must use the **data types** and **protocols** that the server component understands
- These may be defined in a service interface that is separate from the component that does the work, for which there are standard Interface Definition Languages (IDLs), including WSDL

ASIDE: Martin Jewell

- Web Services not = SOA.
- A Web Service with multiple operations in a single WSDL interface is contrary to SOA principles.
- But SOA can be implemented using WSDL if the Web Service is implemented in the right way (singular operation and no polymorphism).

Clients give service instructions, client designers need more

- You tell the waiter you want a menu item and how well cooked it should be.
- A client sends a request **message** to initiate a service**.** This carries the signature of the service – its name and its parameters.

ASIDE

- Client designers need to know more than instructions that make a service run (its signature), before they allow the client to call the service
- The need also to know meaning of the service - what it does with its instructions
  - the functional and non-functional requirements it meets
- And how well the service must do things
  - its non-functional requirements
  - including any commercial agreement covering payment for the execution and maintenance of the service.

A client may or may not wait for a service

- You have to wait for a menu item
  - (You can't go off and do something else)

- A client may use a **synchronous** style and wait for the result,
- or leave a message for the server component to work on.

Every Process reaches an End

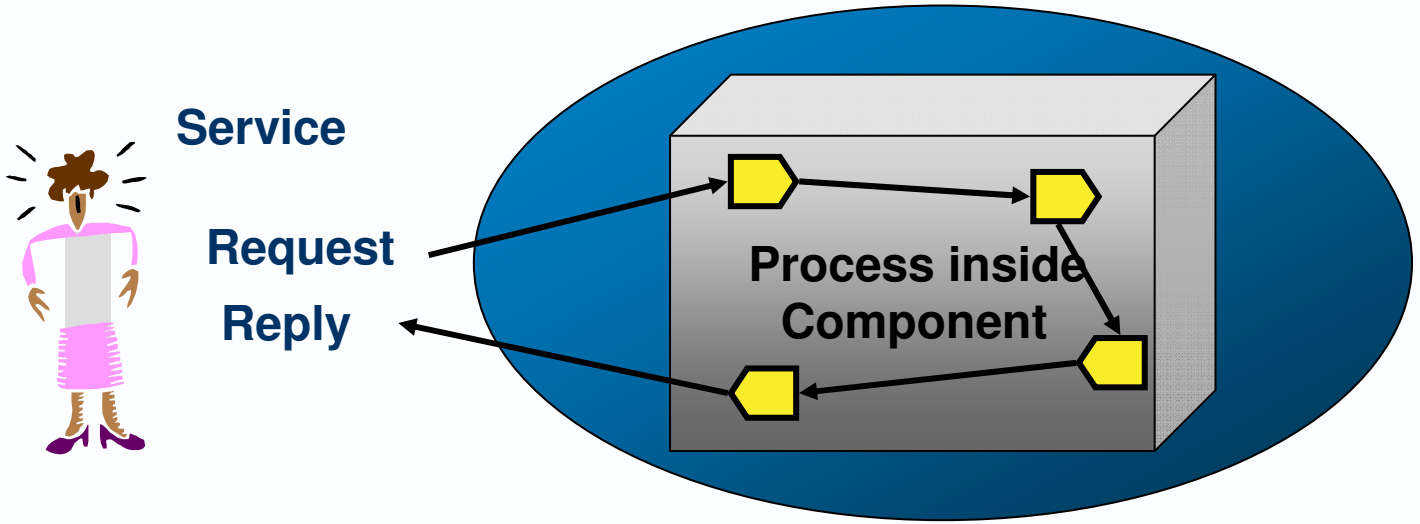
- Every Process ends in a result *
- The result may be called variously
  - The Goal of the Process.
  - The Output of the Process
  - The Service delivered by the Process to one or more consumer Actors.
- * ASIDE
  - In computer science, process is a subtype of procedure.
  - A process is a procedure that terminates - as opposed to a procedure that iterates forever (like the procedure to calculate the value of pi).

Service delivery is the outcome of a process

Avancier

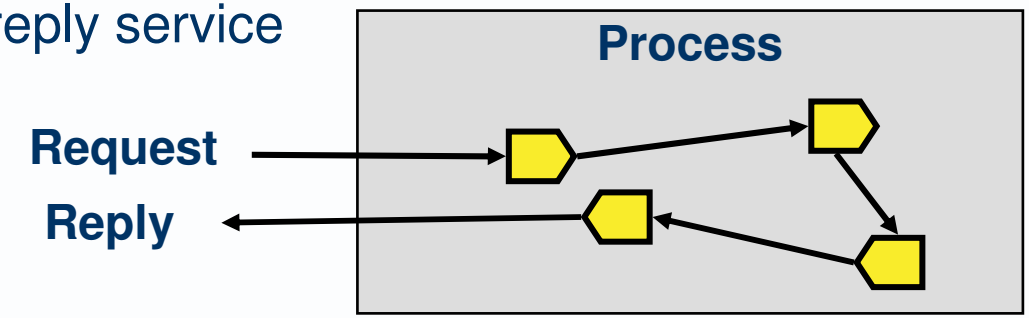
My meal is the end result of a process.

Service delivery is the outcome of process steps executed or orchestrated by a Component

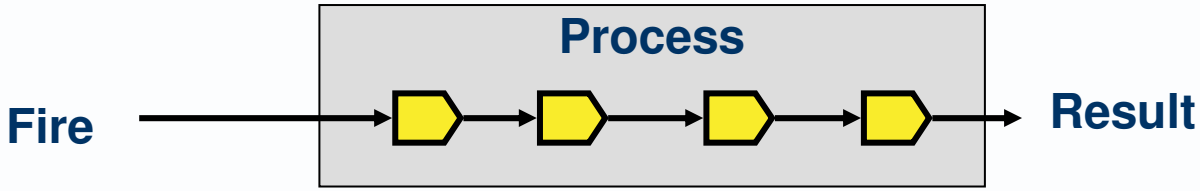


Two kinds of service and process

- Request-reply service



- Fire-and-forget service



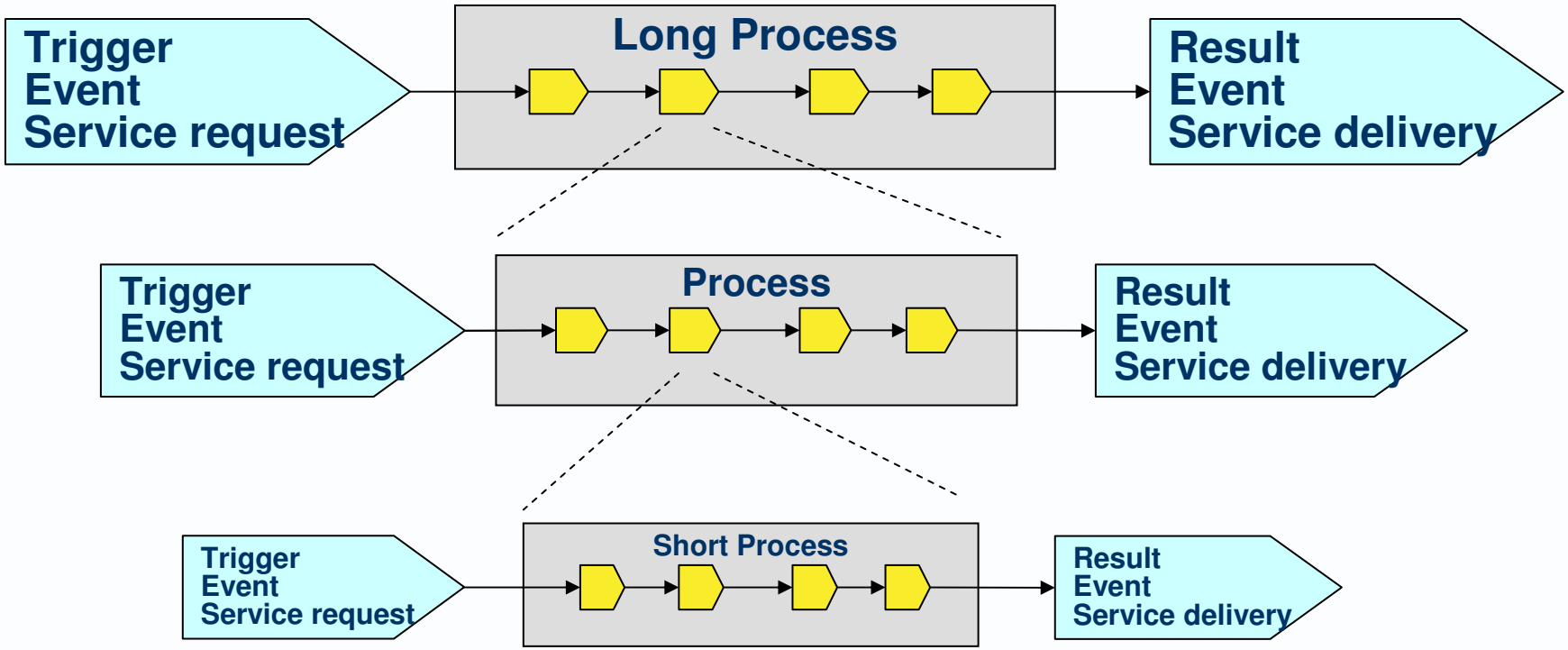
ASIDE: Chris Britton
Most client-server interactions are request-reply or fire and forget, though there are variations.
A technology (e.g. Tuxedo Open/OLTP) might allow a client component to send a message, go off and do something else, but every now and then check to see if a reply has come. Ultimately - once all the other things have been done - the client is waiting for a reply.
A technology might allow a component to initiate parallel threads up and down the client-server stack - talking to a user interface and a data server – e.g. send a message to back to the user's screen if the server is slow, saying "the server hasn't replied yet".

Services and processes are nested

Longer Processes (executed in a wider System)

- Orchestrate

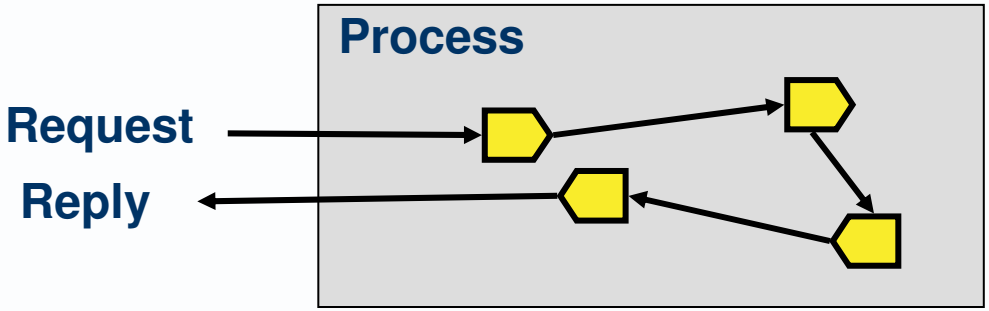
Shorter Processes (executed in narrower Components)



The process may be atomic

Avancier

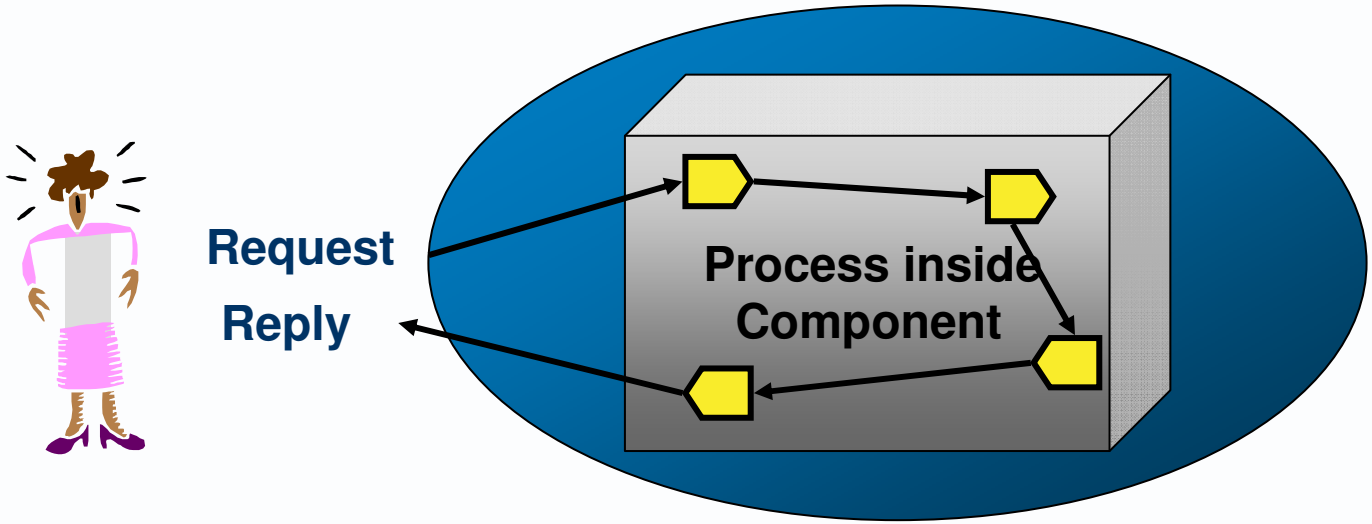
- Half a menu item is unacceptable. You won't pay if it is not complete.
- A service/process might be **transactional**



The process may be contained within a component

Avancier

- A waiter is the only actor to serve a menu item
- A service might be delivered by a process that is executed by one component.



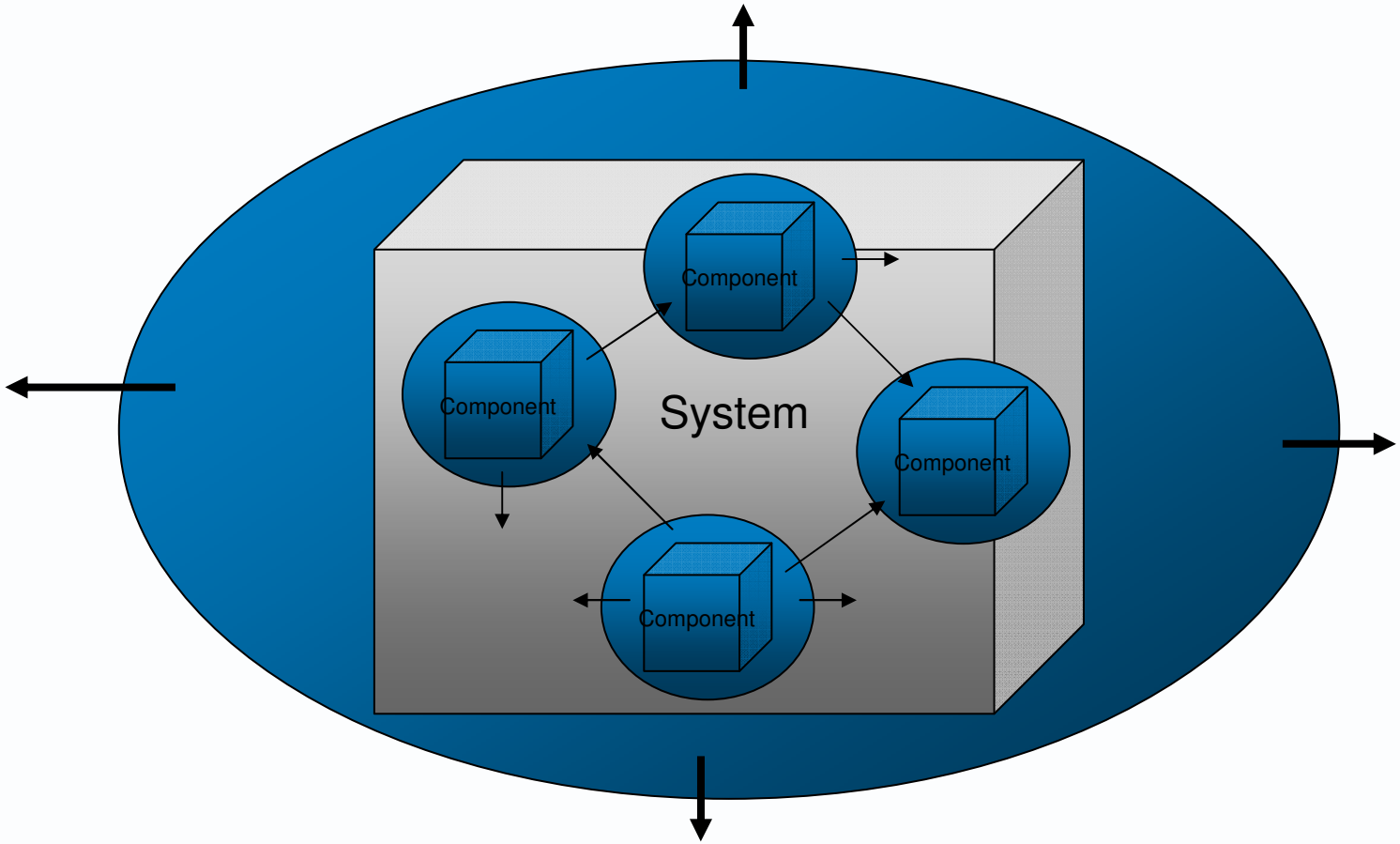
The process may be decomposed and span many components

- A meal is only one step in a longer end-to-end process,
  - You feel hungry
  - You choose a restaurant
  - You schedule an arrival time
  - You order a meal
  - You eat the meal
  - You pay for the service
- Long-term or **end-to-end processes** usually require the cooperation of many components.
- The wider the orchestration, the harder it is to make the process **transactional**.

Components are nested

Avancier

Systems are composed of components, which are composed of components…



Granularity changes how we think about things

- A coarse-grained Component
  - contains and executes finer-grained Processes
- A coarse-grained Process
  - is performed by the cooperation of finer-grained Components

You need a meal

Avancier

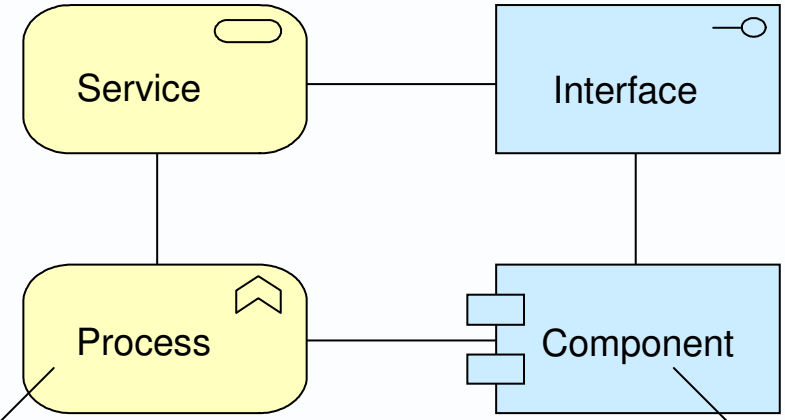
1. You need a meal.
2. Restaurants provide meals.
3. Restaurants publish lists of services (starters, main courses…).
4. Restaurants can publish a service catalogue away from any restaurant
5. You accept one menu item description and price
6. The Restaurant works in a location with the necessary resources.
7. Roads are needed to reach the restaurant.
8. From the restaurant's team of waiters, one must be selected
9. A waiter sits you a table in the restaurant to offer his services.
10. You must talk to the waiter in a language he understands.
11. You tell the waiter you want a menu item, and how well cooked it should be.
12. You have to wait for the service (you can't go off and do something else)
13. My meal is the end result of a process
14. Half a menu item is unacceptable. You won't pay if it is not complete.
15. A waiter is the only actor to serve a menu item
16. The meal is only one step in a longer end-to-end process

Architectural principles in the restaurant story

Avancier

1. Client and Server are roles played by Components
    1. Client components require services
    2. Server components provide services
2. A Service is offered to the terms of a Service Contract
    1. Service contracts are published in interfaces to components
    2. Service contracts may be separated (in directories and facades) from service delivery components
    3. Required service must match provided service
3. Work Places are reached via a Network
    1. Every component needs a work place
    2. A network is needed to reach the work places
    3. A work place may contain several instances (actors) of component type (role)
    4. A server component provides a service via a channel within a work place
4. Clients talk to Servers
    1. Clients and servers must share protocols and language
    2. Clients give service instructions, client designers need more
    3. A client may or may not wait for a service.
5. Service delivery is the outcome of a process
    1. Service delivery is the outcome of process steps executed or orchestrated by a Component
    2. The process may be atomic
    3. The process may be contained within a component
    4. The process may be decomposed and span many components
6. Systems are composed and decomposed
    1. Components are nested
    2. Processes are nested

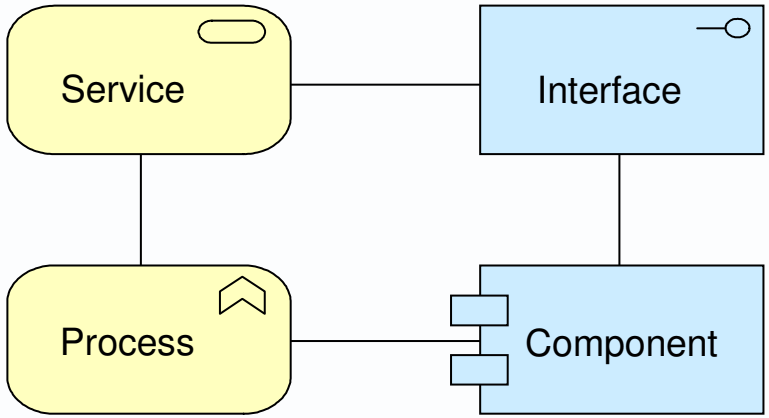
People use “function” ambiguously



An “**Application Function**” is the process in a required use case, which may require one or more application components

A “**Business Function**” is a unit of structure grouping actions deployed to Organisation Units, (which are instantiated and use actors to execute the actions.)

“Interface” is widely used in many way



- Directory
  - A list of services with the addresses of where they can be found
- Façade
  - A channel to a list of services provided by one or more “server” components
- Data flow
  - A message (e.g. a file containing payment records, or a text document) passed from a “sender” component to one or more “receiver” components.
- Protocol
  - One or more layers of protocols needed to invoke services, or send a data flow or via a channel
- Channel
  - A mechanism (telephone, HCI, internet, private network) used to make a client-server connection or transmit a data flow.

ITSM uses words differently again
(definitions below taken from the Common Information Model for ITSM)

Avancier

ITSM uses these words differently

- Process: a single instance of a running program.
  - A user of the OS sees a Process as an application or task.
  - Within an OS, a Process is defined by a workspace of memory resources and environmental settings allocated to it.
  - A Process can execute as multiple Threads which run within the same workspace.
- Service: the availability of functionality that can be managed.
  - may be provided by an entity such as a Logical Device or a Software Feature, or both.
  - typically provides only functionality required for management of itself or the elements it affects.

But uses these words much the same

- System: an entity made of component parts that operates as a 'functional whole'.
  - uniquely named and independently managed in an enterprise.
  - The entire abstraction can be enabled or disabled at a higher level than enabling or disabling of its component parts.
- View: a class providing de-normalized, aggregate representations of managed resources.

A selection of architectural issues dating back a generation

Avancier

- How many levels of granularity to define services, processes, interfaces and components?
- How to design, catalogue and manage those definitions a way that leads to effective reuse*?*
- What to do when we want to a wind back an ill-fated process, but it cannot be made transactional?
- What to do when required services differ *somewhat* from already-provided services?
- At what point does increasing reuse unacceptably raise
  - Complexity of configuration management?
  - Costs of maintenance?
  - Pressure on service level agreements?

- These questions predate SOA, and will postdate it

This presentation is for those interested in aligning different architecture frameworks

1 of 6 related presentations in the Library at http://avancier.co.uk

Avancier

Logicality

Process threads you will find in various architecture frameworks

Modularity

Foundation concepts and strands in the modelling of human and computer activity systems

Granularity

The challenge of multi-level goals, plans and specifications

Architecture meta meta concepts

A 4 cell schema for modelling systems, which helps you understand meta models

Functionality

Functions, Organisation Units and Processes in human activity systems

Architecture meta models

Comparing the meta models of industry standard architecture frameworks