Génie logiciel

Philippe Dugerdil

05.12.2019

Architecting distributed service apps

Now we understand the plumbing, but
how to design a distributed system ?

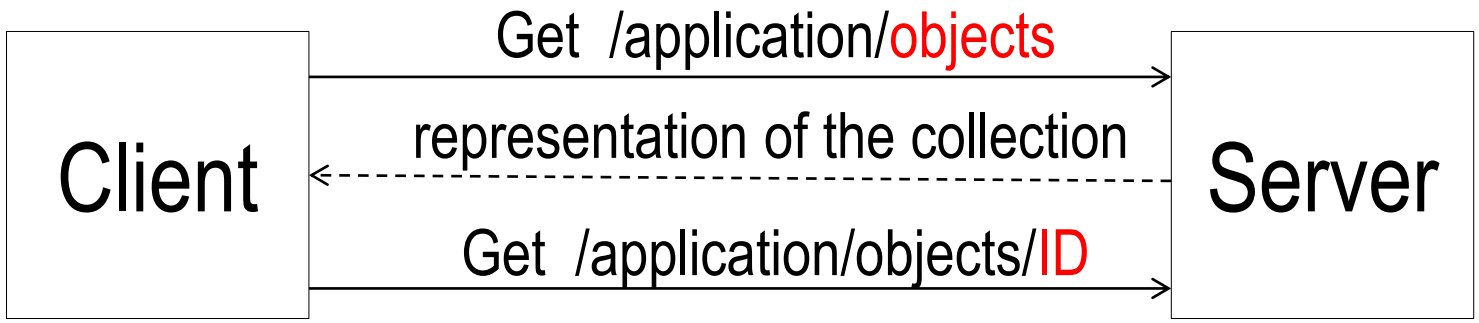
A few patterns…

Collection Pattern

- **Problem**: how to select objects among a set of objects to pass them further to another service ?
  - For bandwidth reasons, one cannot pass ALL objects to the client for it to select one of them.

- **Solution**: create a collection of representations of objects for selection purpose. Contents:
  - Meaningful description for selection
  - Unique ID

Collection

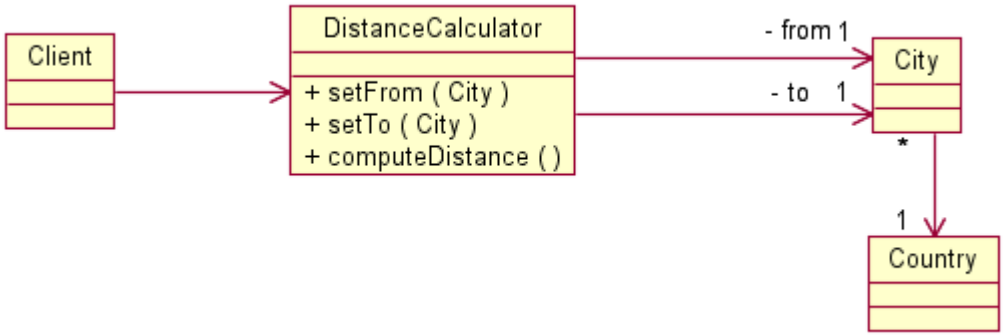


Function as a Resource pattern

- **Problem**: the service to implement is not the access to a business object but the computation of a function.
  - What resource should it be attached to ?
- **Solution**: create a resource (URL) to represent the function itself

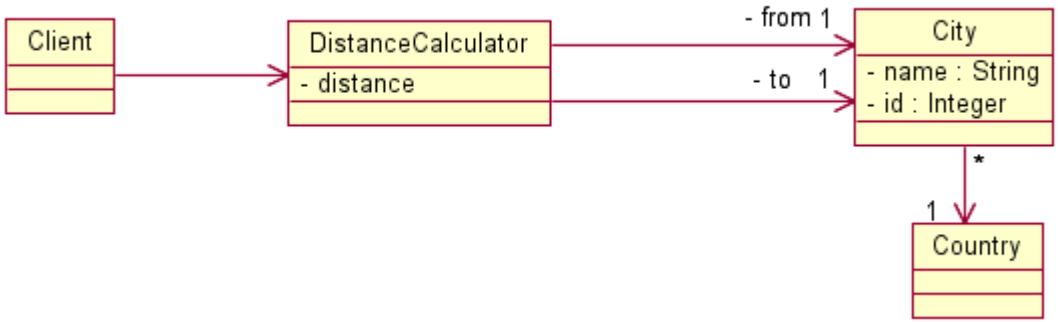
Example

- Computing the distance between 2 cities
  - If we did an OO analysis we would have designed a "service provider" object to handle the responsibility



  - In WS design we would do the same, but with a GET query

Function as a resource



GET /applic/distanceCalculator/distance?from=id1&to=id2

```
@Path("/distanceCalculator/distance")
@GET
public String computeDistance(@QueryParams("from") String fromId,
                              @QueryParams("to") String toId) {…}
```

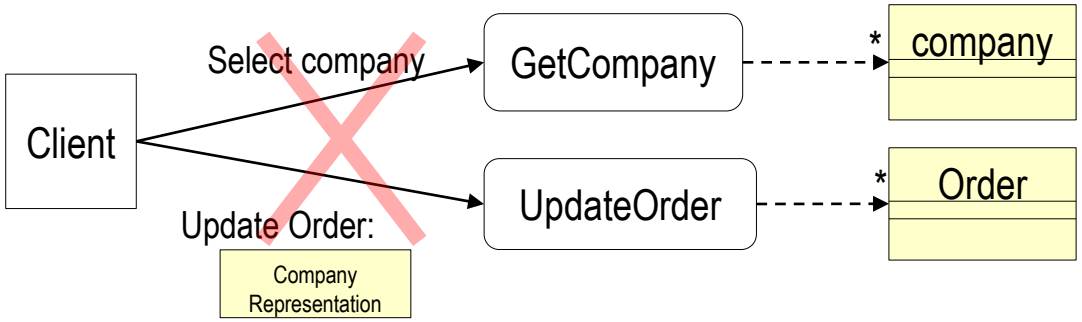
How to get the city Ids? : **Collection** pattern

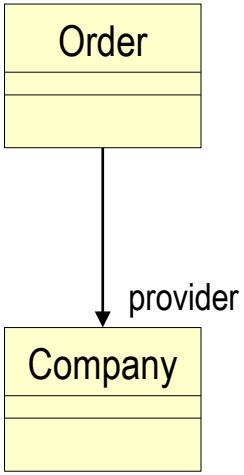
Use references (BMS)

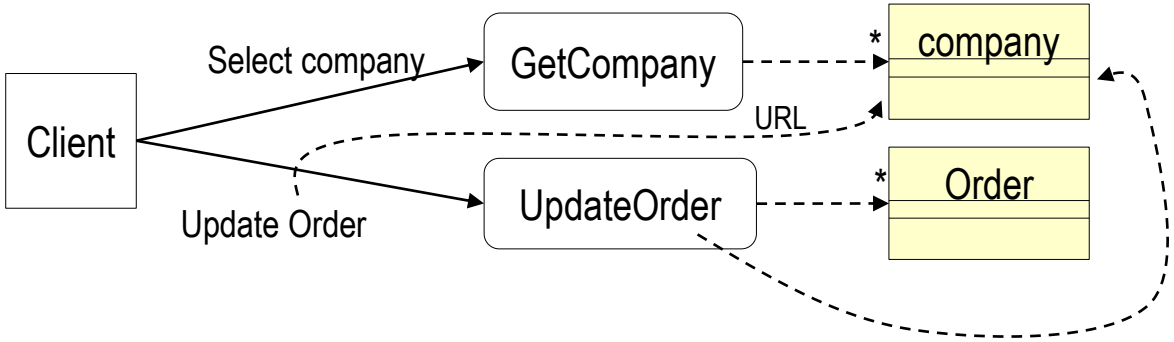
- **Problem**: a service processes some resource's information.
  - How should the client of the service pass this information to another service so that it minimizes the chance for the information to be obsolete?

- **Solution**: if the structure of the resource is known to the second service, then pass a link.
  - Then it will be up to the second service to retrieve the information at the time it is needed.

Use reference in pictures







Request / Acknowledge

Problem: how to handle long lasting services i.e. services that cannot reply immediately?

Solutions:

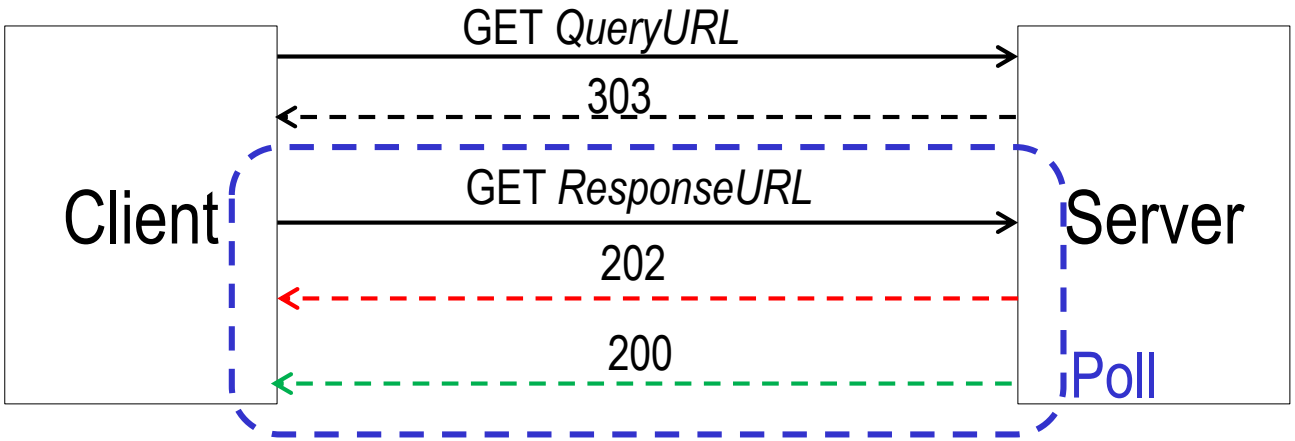
- Request/acknowledge/poll
- Request/acknowledge/callback

Request/acknowledge/poll

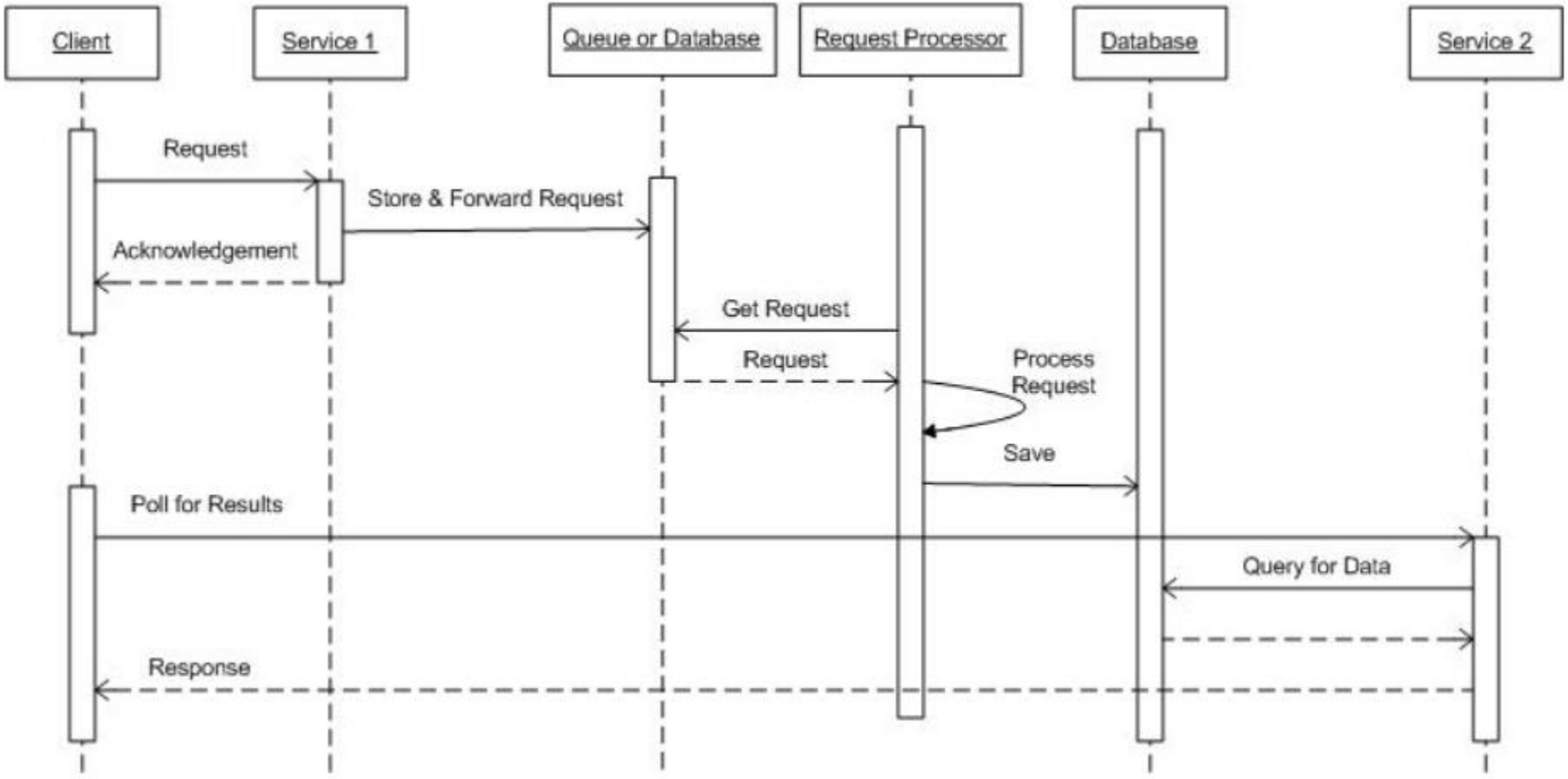
The query, once submitted is returned immediately

- Return code: 303 (see other) & provision of a new URL (**Location**).
- The client will poll the provided new URL. Return codes:
  - 202 Accepted, as long as not ready
  - 200 OK, when ready

Since the response is provided through another URL there is no overlap



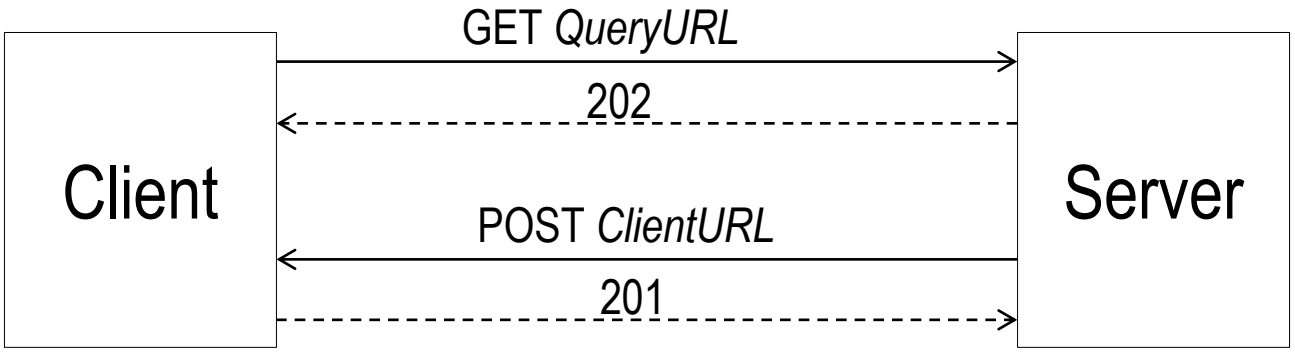
Interactions



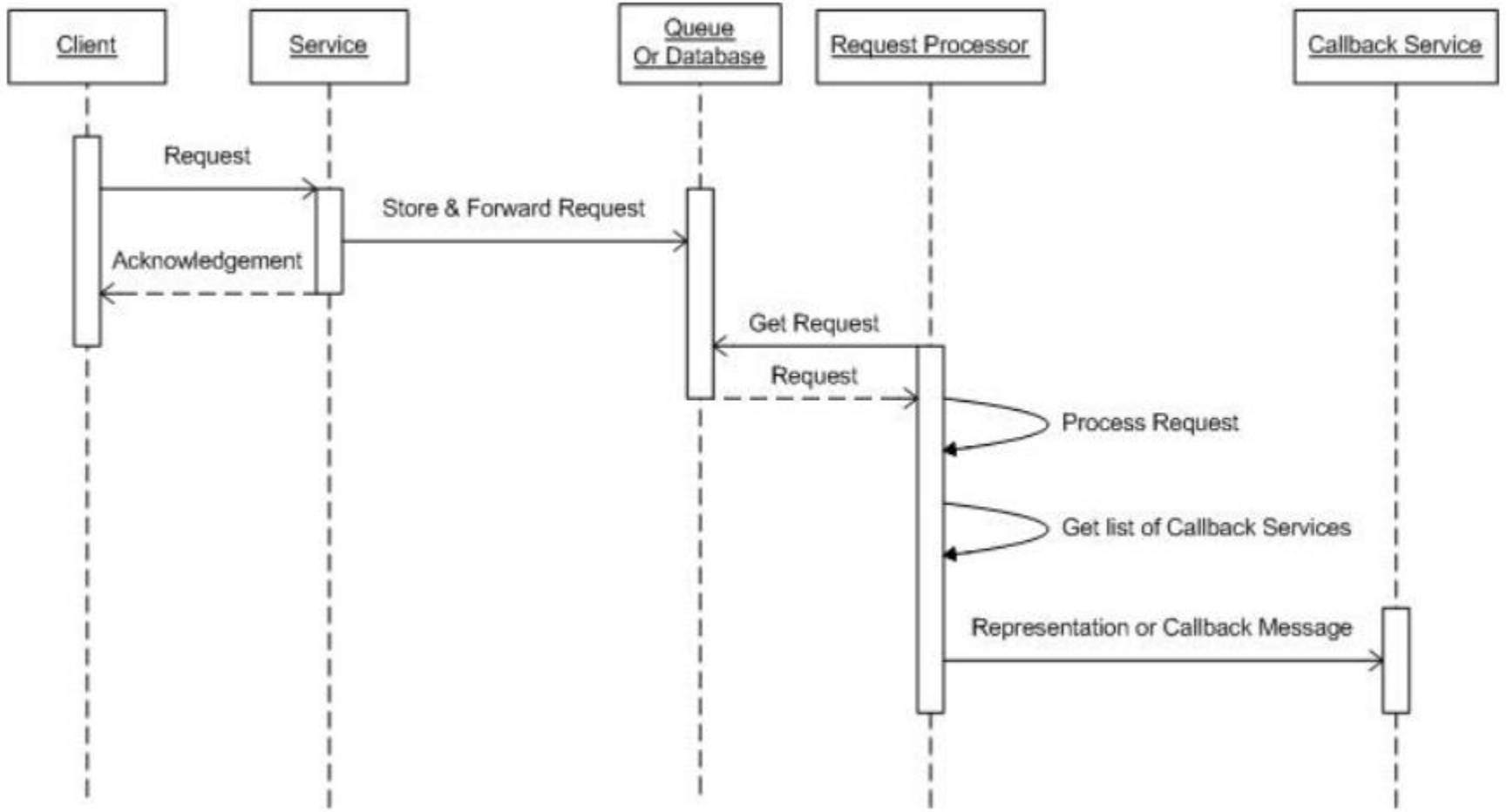
Source: http://servicedesignpatterns.com/clientserviceinteractions/requestacknowledge, accessed 2017.05.07

Request/acknowledge/callback

- The GET query contains the URL to which the response must be sent. The query is returned immediately with 202 (Accepted)
- When the response is ready, the server sends it to the recipients through a POST (PUT) query to the identified resources (which return 201 (200))
  - In the general case (several recipients), the *payload* includes the URL of the client resource to be updated when the processing completes
  - If the callback URL is unique, it can be passed in the location header



Interations



Source: http://servicedesignpatterns.com/clientserviceinteractions/requestacknowledge, accessed 2017.05.07

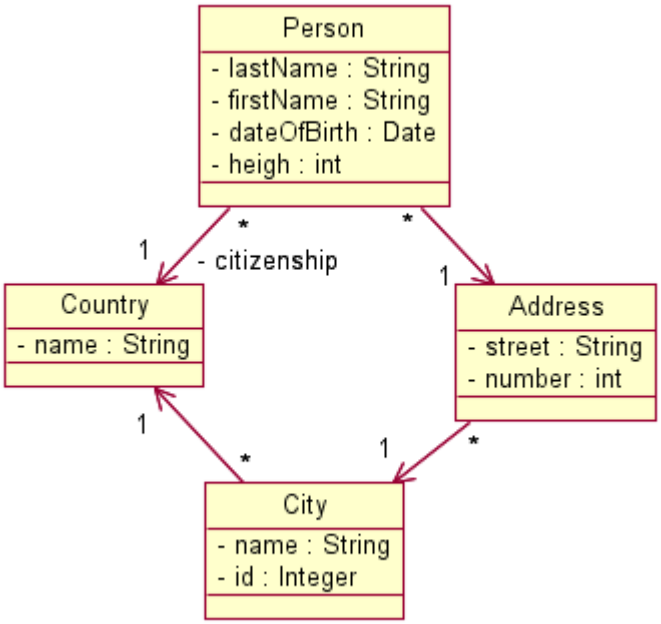
Pattern
Data Transfer Objects

- Problem: to minimize network traffic, one does not want to send/receive all information of domain object, but only the meaningful part for the service
  - How to extract and the selected part of the domain object for network transfer. Constraint: one does not want to implement transfer-specific code in domain objects

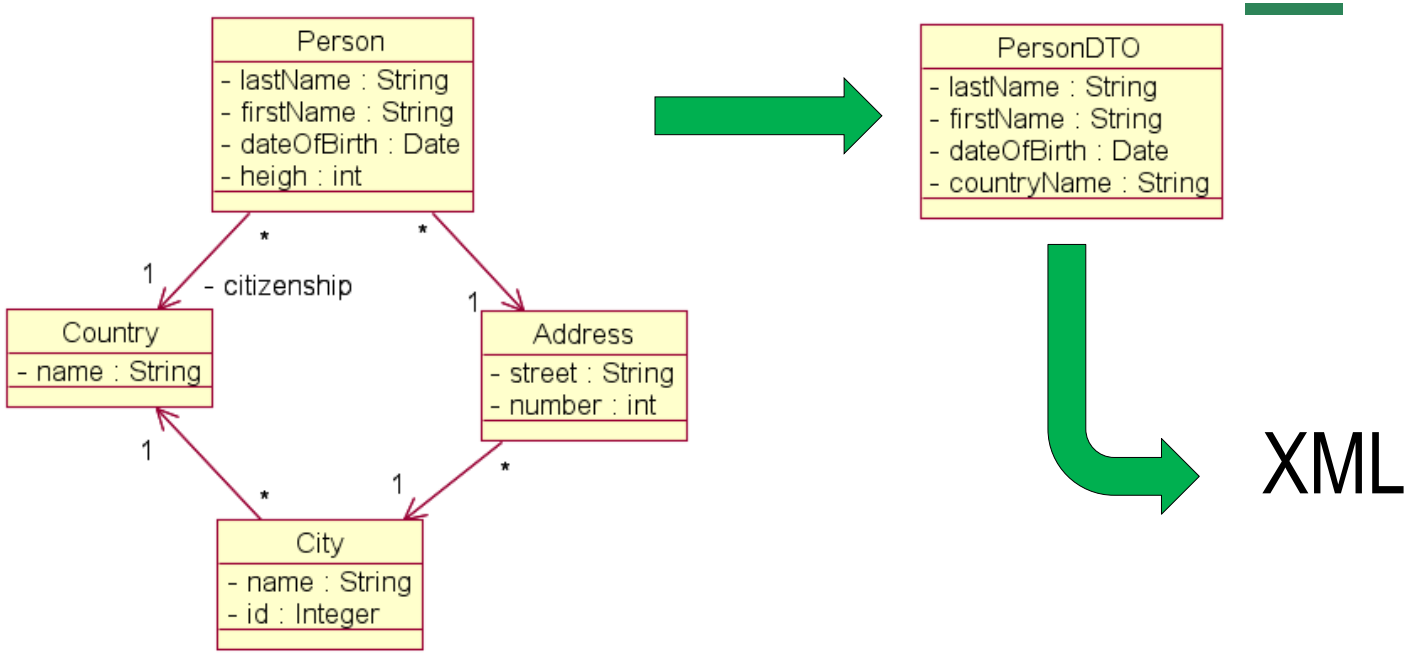
DTO

Solution: create a Data Transfer Object to contain only the part of the domain object we must transfer (or parts of several objects).

Example: when translating some Person data (name, birthdate, country) to XML we do not want to include the full object tree.



Solution DTO

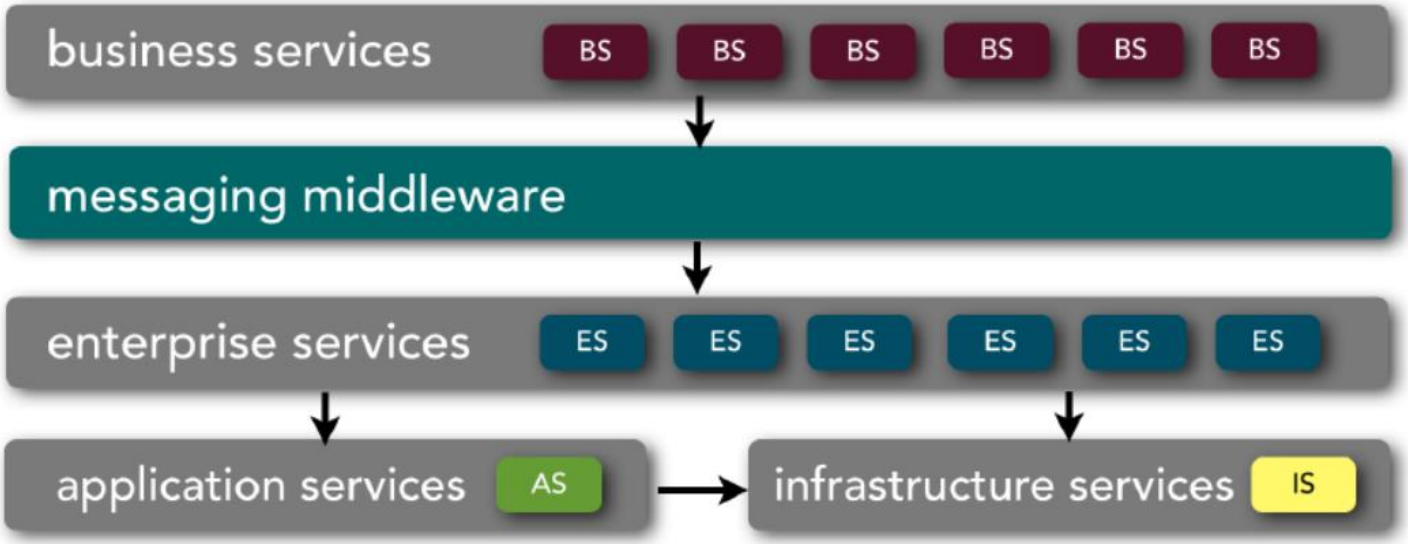


- The DTO object can easily be translated to XML using JAXB, without impacting the domain object
- No being a business object, the DTO can be annotated

Services composition

Reminder: SOA

- Attempt to architect large systems made from services
- Enterprise services were intended to be shared across the organization
- Enterprise architecture mindset when designing the system



[Richards M. - Microservices vs Service Oriented Architecture. O'Reilly, 2016]

SOA

- Service components can range in size anywhere from small application services to very large enterprise services.

- It is common to have a service component within SOA represented by a large product or even a subsystem.

[Richards M. - Microservices vs Service Oriented Architecture. O'Reilly, 2016]

So what's next?

Microservices!

What is microservice

- Microservices is a specialization of an implementation approach for service-oriented architectures (SOA) used to build flexible, independently deployable software systems.

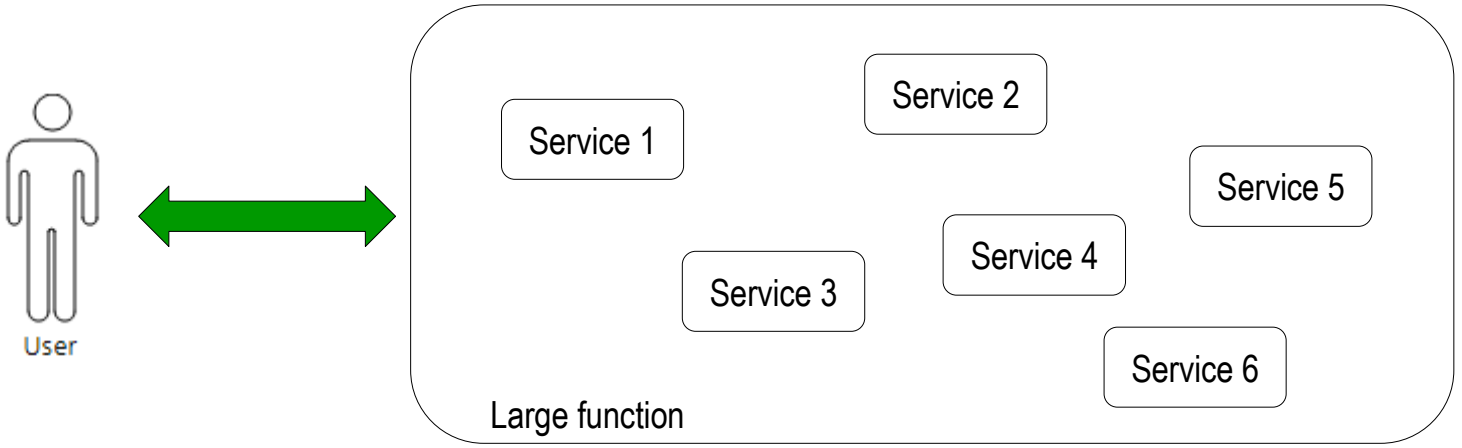
  [Wikipedia]

- The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

[https://martinfowler.com/articles/microservices.html#CharacteristicsOfAMicroserviceArchitecture]

Constraint: large systems

- **Services** are used either in isolation or in groups to perform some larger task (composition)



New QA's for a new world

- Scalability
- Independance
- Deployability
- Changeability

(~~ACID~~)

QAs

- Scalability
  - If one of the feature of the program is heavily used, one should be able to scale up the resources
- Independence
  - Services should not interact with each other. A change in one service should not have an impact on other services
- Deployability
  - Services must be independently deployable on different VMs
- Changeability
  - Services should be easily changeable without impacting the rest of the system

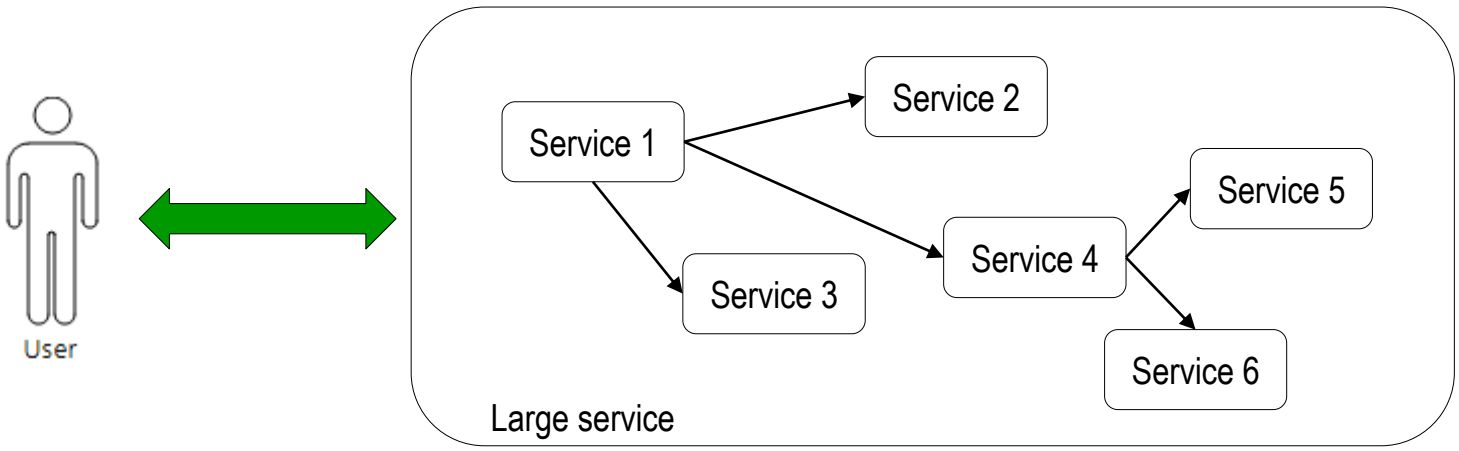
Each Microservice is independently called through the network

API: Message oriented (REST)

Architecting large systems

We must build large services (business functions) from simpler ones

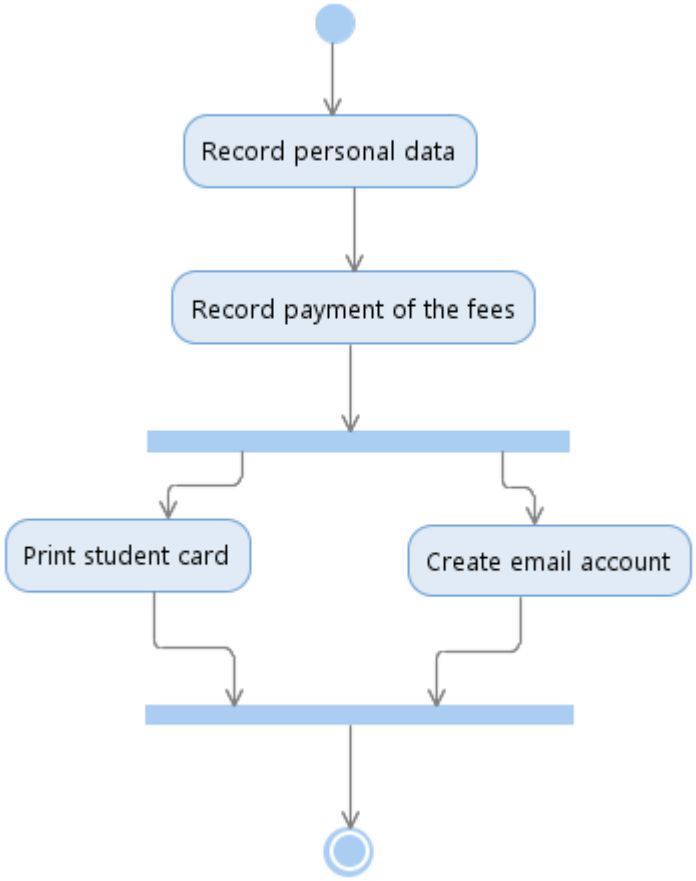
- What are the possible calling architectures ?



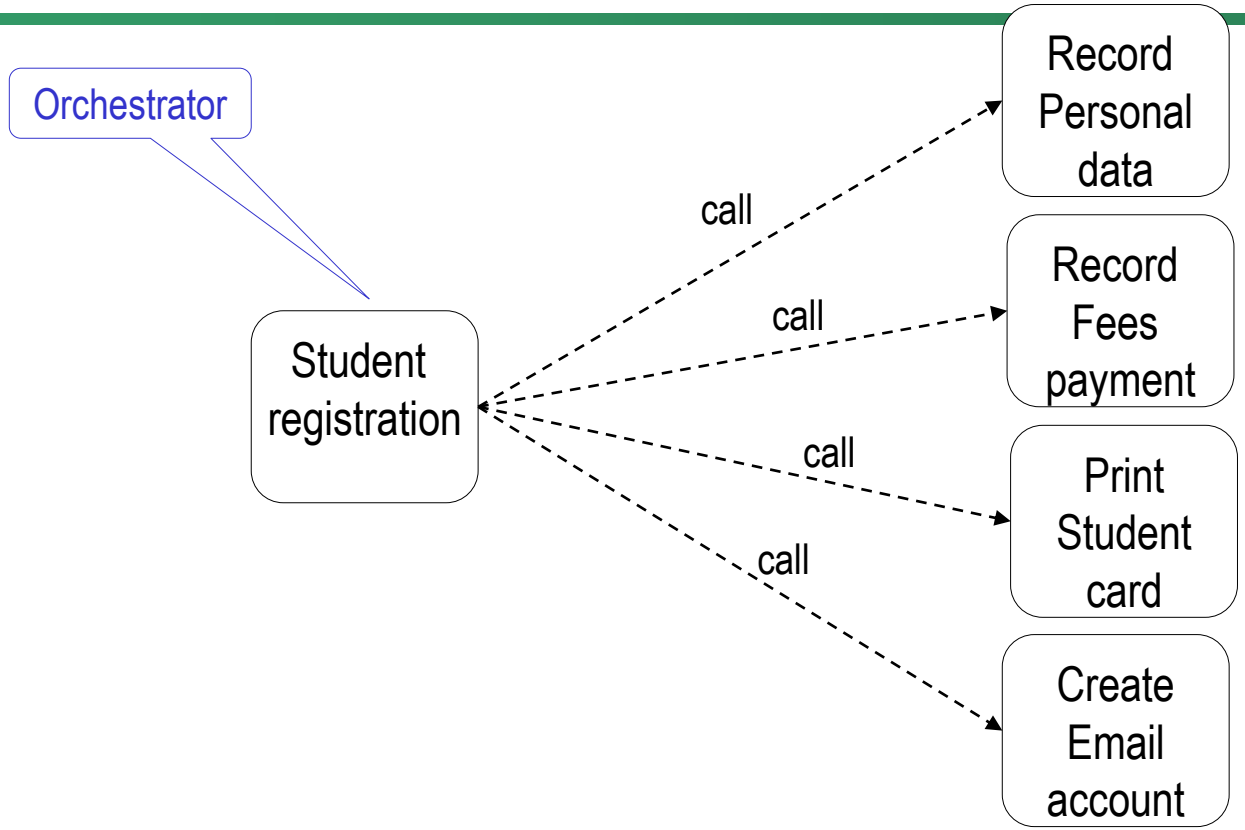
Calling architecture: Orchestration and Choreography

- Orchestration
  - A manager orchestrates the work of the other services to implement the business function /process
    - Each service is explicitly called by the manager
- Choreography
  - Publish/subscribe kind of architecture : the scheduling is implicit in the sequence of event generated by each service.
    - Each service registers to the event it must process.

Example of a simple business process: enrollment of students

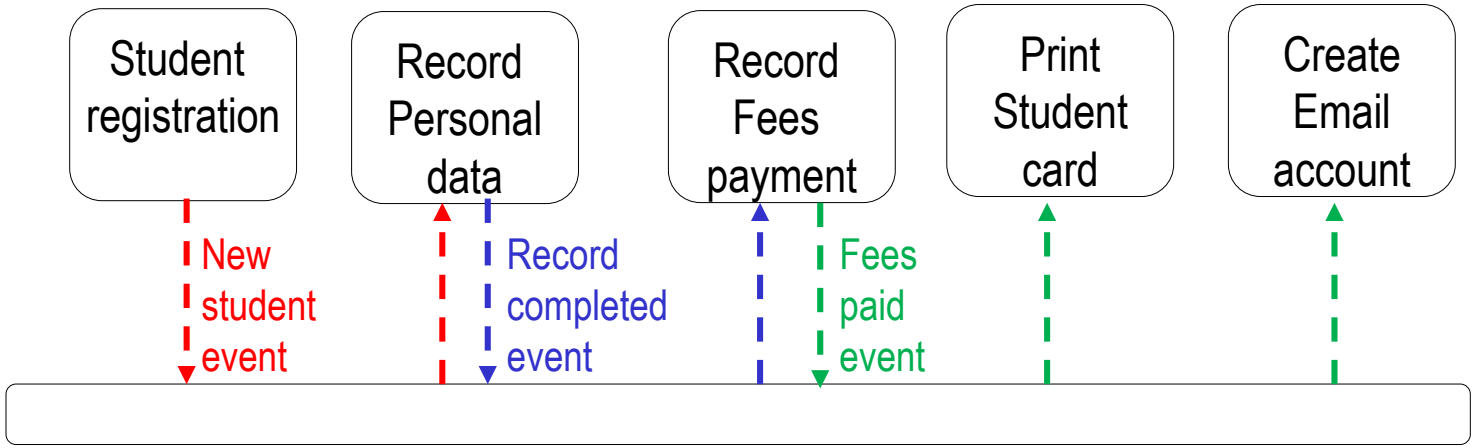


Orchestration



- Good Auditability: central monitoring of the subtask's completion
- Extensibility: to call new services, we must adapt the orchestrator

Choreography



The services must first subscribe to the proper events
(publish/subscribe architecture)

- Auditability: harder to monitor the subtask's completion
- Good Extensibility: simple registration of the new component to the bus

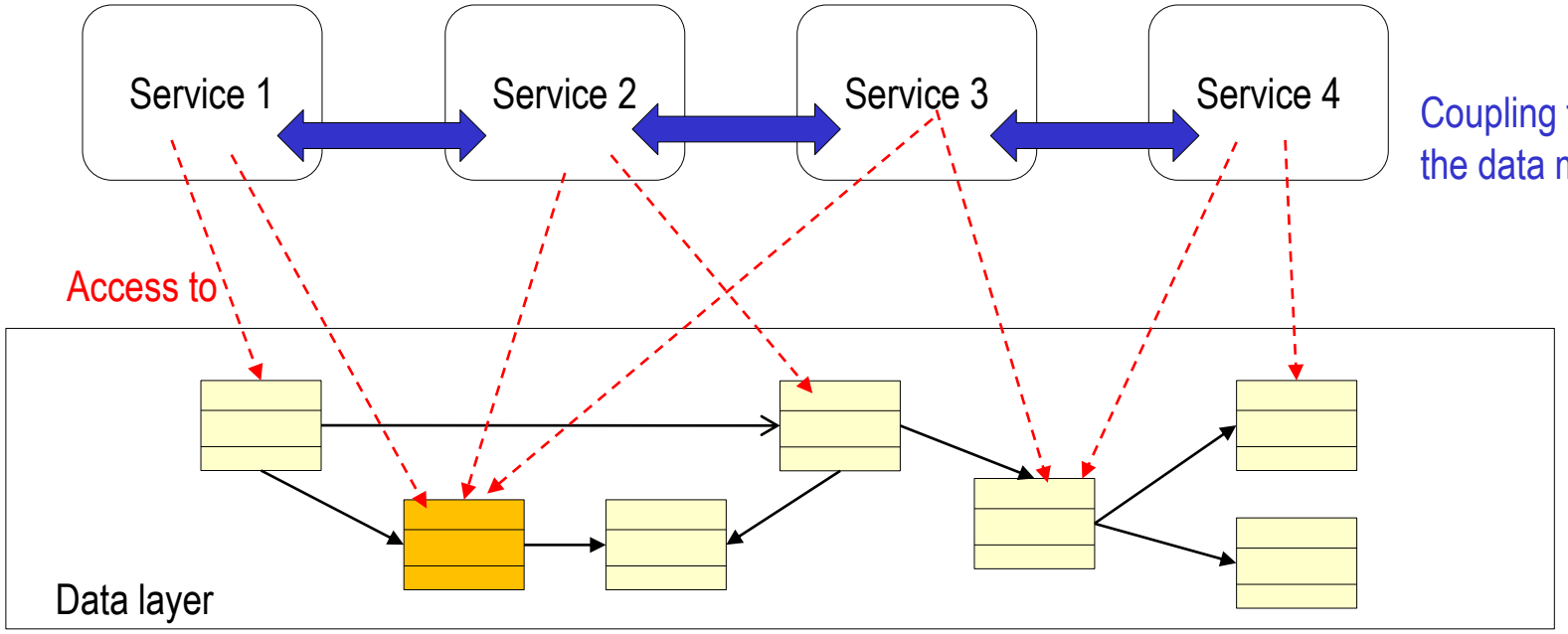
Designing the microservices

Traditional approach to system design

- Design of the data model
- Build the services that manage each part of the model

- But this will not comply with the independence & deployability QAs
  - The services will be linked through the data model
  - The service cannot be independently deployed

Example of drawbacks



- ~~Independence~~
- ~~Scalability~~
- ~~Deployability~~

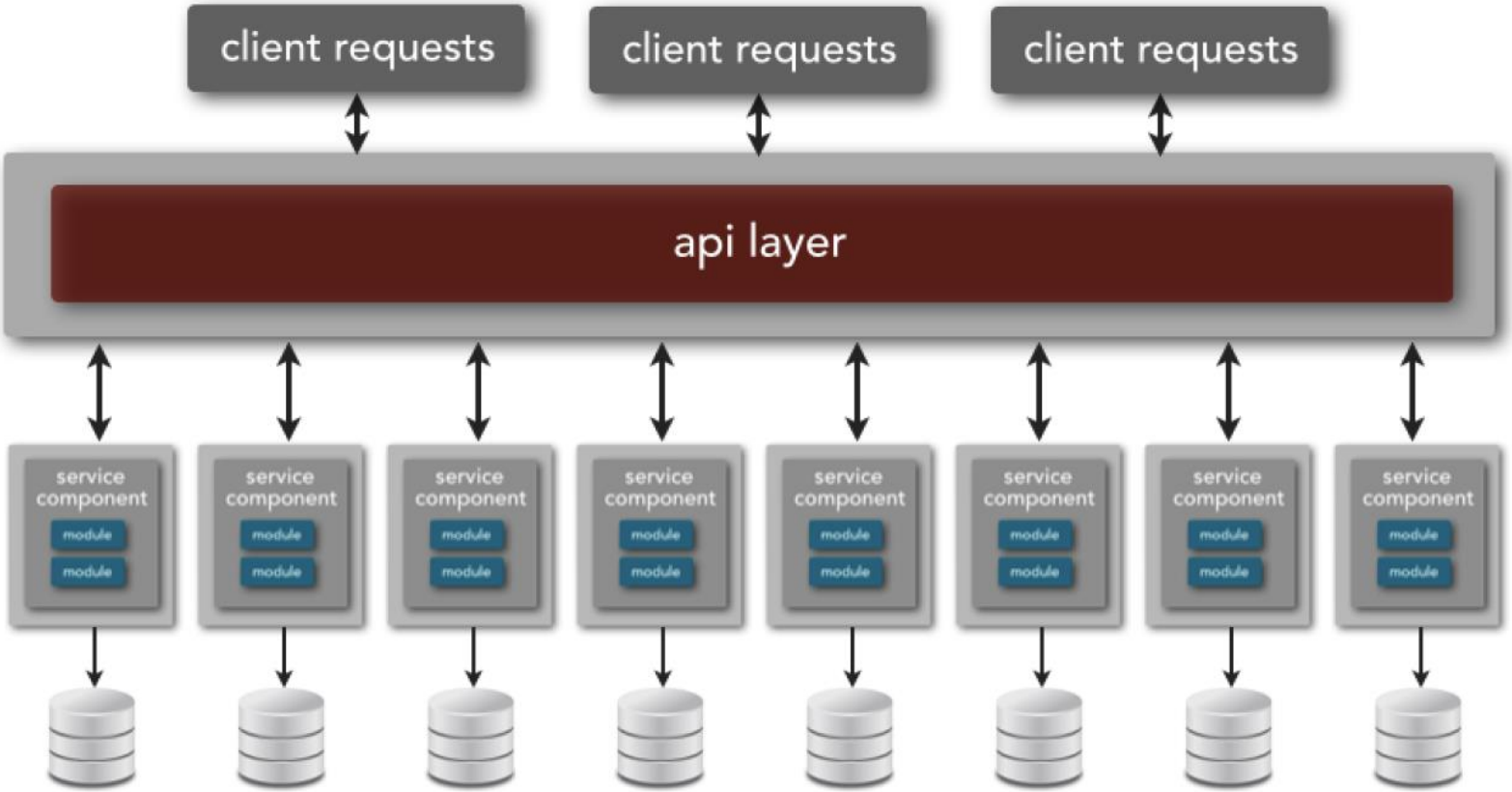
Consequences

- Independence
  - If the part of the data model required by service 1 is changed (dark yellow class) there will be an impact on other services
- Deployability
  - Services are strongly linked to the data layer. How could they be independently deployed on several VMs?

Learning

- Avoid data sharing among services for large systems based on microservices

Avoiding data dependence among services

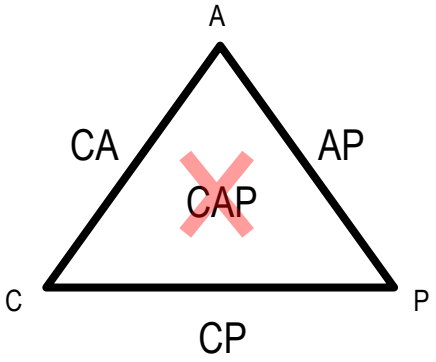


Source: Richards M. - Microservices vs Service Oriented Architecture. O'Reilly, 2016

CAP theorem

Il est impossible sur un système informatique de calcul distribué de garantir en même temps:

- La Consistance des données) (**Consistency**)
- La Disponibilité (**Availability**)
- La tolérance aux Partitionnement (**Partition** Tolerance) : le système doit fonctionner même s'il est partitionné sur plusieurs nœuds



Sources:

- https://dzone.com/articles/better-explaining-cap-theorem
- wikipedia

C'est ce que nous observons avec les microservices sur des machines distribuées sur un réseau

Findings about distributed data management

1. Many clients query the data while a few actually update it
   - Scaling happens mainly on the read service
   - But several updates may happen simultaneously.

2. The data people watch are not guaranteed to be up to date
   - When someone retrieves data, someone else may have updated it simultaneously. So data read may be partially obsolete. This is a fact in large distributed systems
     - Do not over engineer a solution to avoid obsolete data be displayed. The impact on QA will be too heavy

Conclusion

1. All the services do not need the same access to the data (read / write)

2. Read services may need to display several parts of the data model while write services generally focus on a single one

3. The “Permanent data consistency” * constraint must be released for large distributed systems (CAP Theorem)

*All data being consistent across all clients at all times