# UE AD FIL A1 - RPC et gRPC

*2023-2024*

**Hélène Coullon**

**IMT Atlantique**
Bretagne-Pays de la Loire
École Mines-Télécom

# Table of Contents

# Les concepts de base de RPC

# RPC

Remote Procedure Call

1. Définition de l'API avec un "Interface Definition Language" (IDL)
2. Appel à distance par un client d'une méthode/procédure fournie par un serveur
   - la *distance* est cachée à l'utilisateur
   - utilisation des méthodes du serveur comme une bibliothèque
   - fonctionnement basé sur un design pattern de Proxy

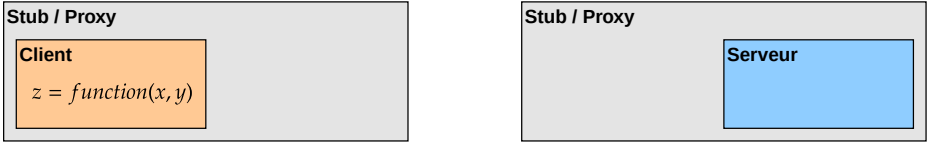
# Interface Definition Language (IDL)

- Langage pour définir une interface, un contrat entre un client et un serveur (entre des services)
- Permet de définir la signature des méthodes/procédures distantes
- Permet de spécifier le type des messages d'entrée et de sortie des méthodes

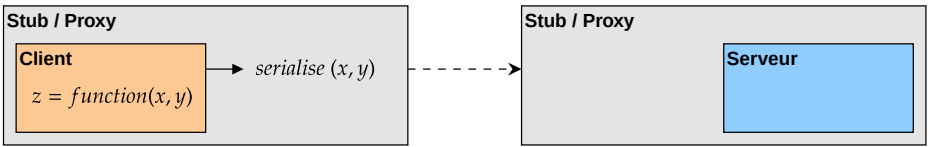
# Proxy et “Stub”

- Les stubs sont les codes générés automatiquement depuis la définition de l’interface avec l’IDL
- Il s’agit de deux proxy : un pour le client et un pour le serveur
  - sérialisation/désérialisation des entrées/sorties
  - aussi appelé marshalling/unmarshalling
  - transformation des données de la mémoire locale pour envoi sur le réseau
  - réception et transformation des données reçues par le réseau
  - gestion de protocoles réseaux utilisés

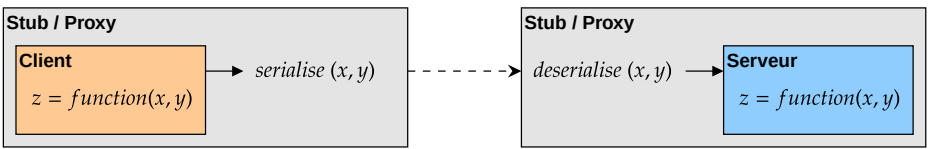
# Diagramme de fonctionnement



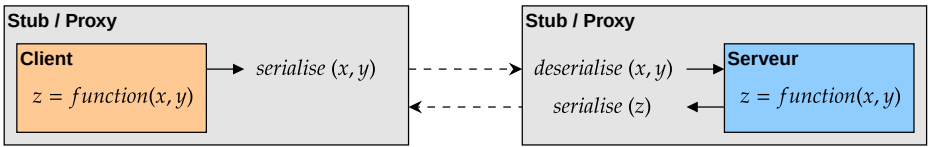
# Diagramme de fonctionnement



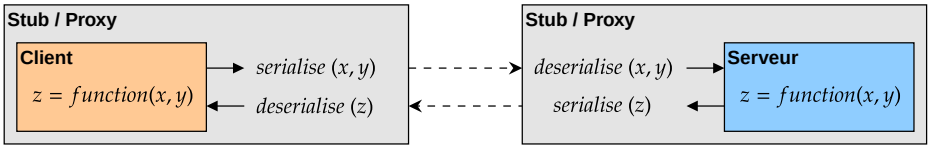
# Diagramme de fonctionnement



# Diagramme de fonctionnement



# Diagramme de fonctionnement



# Un peu d'histoire

# RPC

Remote Procedure Call : ce type de communication et d'API entre un client et un serveur est apparu avant REST

- Apparition dans les années 1980

## CORBA 1991

- Common Object Request Broker Architecture
- Basé sur TCP/IP
- Composants sur des OS différents
- Composants écrits en différents langages
- Composants sur des réseaux différents

## SOAP 1998

- Simple Object Access Protocol, Microsoft
- Standard d'échange entre "web services" utilisant le format XML
- Basé sur HTTP (portabilité)
- SOAP est le successeur de XML-RPC
- l'IDL de SOAP est "Web Service Definition Language" (WSDL)
- SOAP est toujours réputé pour sa sécurité et largement utilisé pour les systèmes bancaires !

Un premier pas vers REST avec SOAP !

# gRPC

# De retour dans les années 90 ?

- Framework OpenSource proposé par Google
- Basé sur HTTP/2.0 pour allier flexibilité et efficacité
- Compatible avec de nombreux langages modernes
  - https://grpc.io/docs/languages/
- Utilisé par des acteurs majeurs
  - Google (bien sûr)
  - NetFlix
  - CoreOS
  - CISCO
  - etc.

# Protocol Buffers - l'IDL de gRPC

- Spécification des signatures de procédures par service
- Structure des messages d'entrée et de sortie

```
syntax = "proto3";

service Movie {
    rpc GetMovieByID(MovieID) returns (MovieData) {}
    rpc DeleteMovieByID(MovieID) returns (MovieData) {}
    rpc UpdateMovieByID(MovieData) returns (MovieData) {}
    rpc GetMovieByTitle(MovieTitle) returns (MovieData) {}
    rpc GetListMovies(Empty) returns (stream MovieData) {}
}

message MovieID {
    string id = 1;
}

message MovieTitle {
    string title = 1;
}

message MovieData {
    string title = 1;
    float rating = 2;
    string director = 3;
    string id = 4;
}

message Empty {

}
```

# Les types de procédures

- Unary RPCs
  - Le client envoie une seule requête au serveur
  - Le client reçoit une unique réponse
  - Appel de fonction normal
- Server streaming RPCs
  - Le client envoie une seule requête
  - Le client reçoit une séquence de messages (stream) en réponse
- Client streaming RPCs
  - Le clilent envoie une séquence de messages (stream)
  - Le client reçoit une unique réponse
- Bidirectional streaming RPCs
  - Le client envoie un stream
  - Le client reçoit un stream

# Les types de procédures - Exemples

**Unary RPC**

```
rpc Hello (HelloRequest) returns (HelloReply)
```

**Server streaming RPC**

```
rpc Hello (HelloRequest) returns (stream HelloReply)
```

**Client streaming RPC**

```
rpc Hello (stream HelloRequest) returns (HelloReply)
```

**Bidirectional streaming RPCs**

```
rpc Hello (stream HelloRequest) returns (stream HelloReply)
```

# Appels synchrones et asynchrones

- Synchrone : l'appel de méthode est bloquant
- Asynchrone : l'appel de méthode est non bloquant, une synchronisation est rendue explicite quand la réponse est nécessaire

# Stub et Servicer générés

## Deux fichiers générés à partir de l'interface `nom.proto`

- `nom_pb2_grpc.py` : contient les stubs client et serveur
- `nom_pb2.py` : gère les types de messages et la sérialisation/désérialisation

## Dans `nom_pb2_grpc.py` un Stub et un Servicer

- Stub
  - Il s'agit du Stub client
  - Le code client doit utiliser cette classe pour faire les appels distants
- Servicer
  - Il s'agit du Stub serveur
  - Le code serveur doit surcharger cette classe pour donner l'implémentation des procédures distantes

# Comparaison à REST et GraphQL

| | Performances | Debug | Découverte | Flexibilité | Nb fonctions |
|---|---|---|---|---|---|
| REST/OpenAPI | | | | | |
| gRPC | | | | | |
| GraphQL | | | | | |

| | gRPC | REST/OpenAPI |
|---|---|---|
| Flexibilité | modifier fichier proto<br>re-générer Stubs<br>modifier les codes | modifier requêtes<br>modifier les codes |
| Dépendances | install gRPC pour tous les langages | installation bibliothèques |
| Travail | explosion du nb procédures | explosion nb points d'entrée |

# Exemple d'application mélangeant gRPC et REST

Un exemple d'application fournie par Google qui utilise gRPC pour les communications entre les micro-services internes, avec plusieurs langages à la fois.

https://github.com/GoogleCloudPlatform/microservices-demo

# Tutoriel

Tutoriel gRPC