Architectures logicielles

1. Architecture et styles architecturaux

1.1 Travail à réaliser - En binôme

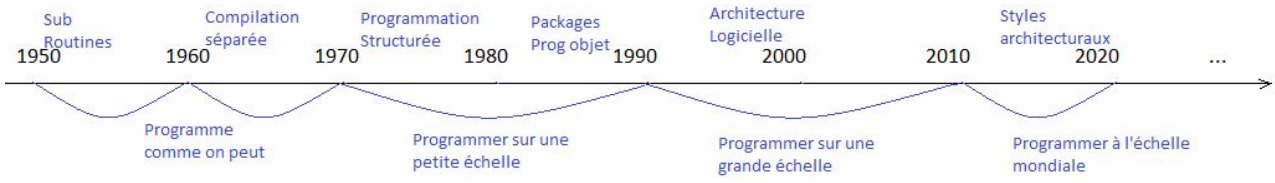
Ecrire un rapport d'une page et demi à deux pages sur le Bitcoin et les Blockchain :
- Définir les deux termes,
- Définir le lien et l'architecture entre les deux,
- Montrer qu'on a compris le principe de fonctionnement,
- Expliquer pourquoi ceci a été créé, donner des pour (se libérer des banques centrales) et des contre afin de nuancer les propos

1.2 Introduction

1.2.1 Historique

Sub routines (assembleur)	1950	Programmer comme on peut
Compilation séparée (Pascal, Cobol)	1960	
Programmation structurée	1970	Programmer sur une petite échelle
Packages, programmation objet Père de la programmation objet : Minsky, pour avoir un langage plus naturel pour les Hommes. Programme objet : Ensemble d'objets qui communiquent entre eux afin de résoudre un problème.	1980	
	1990	Programmer sur une grande échelle
Architectures logicielles	2000	
Styles architecturaux Objectifs : Faire qu'il y ait interopérabilité entre les styles architecturaux	2010	Programmer à l'échelle mondiale
	2020	

Vision historique de l'architecture logicielle



Vision historique du tableau

<u>Programmation kilomètre</u>
Programmer chaque fonction qui arrive ⇒ le programme est fonctionnel mais devient illisible

Programmation structurelle

Début du typage, on peut séparer le travail par équipe

Programmation objet :

Nous vient de Marvin Minsky, avant programmation = fonction, Minsky disait que ce n'était pas naturel, qu'un humain ne pense pas de cette façon (*exemple du ciné : on sait qu'on est au ciné grâce à la reconnaissance d'objets, après on recherche l'utilité du bâtiment*).

La programmation objet est une métaphore de la société, des objets qui communiquent entre eux et qui permettent de résoudre des problèmes

1.2.2 Définitions

Définition d'une architecture logicielle

Une architecture logicielle est définie par un ensemble de composants (objets, BDD, serveurs, ...) ainsi que par la description des interactions entre ceux-ci (appels de procédures, envois de messages, émission d'événements, ...). Elle permet de spécifier les caractéristiques d'un système en définissant :

- Sa structure : éléments de traitements et leurs interactions,
- Son comportement : fonctionnalités et protocoles de communication,
- Ses propriétés globales (sécurité, vivacité, ...)

Archi = { Éléments (de structure ou de comportement), formes (relations et propriétés entre les éléments), raisonnement (raisonnement à faire sur cette architecture donc sur l'ensemble des éléments) }

Qu'apportent les architectures logicielles ?

Les architectures logicielles jouent un rôle important dans au moins six aspects du développement logiciel :

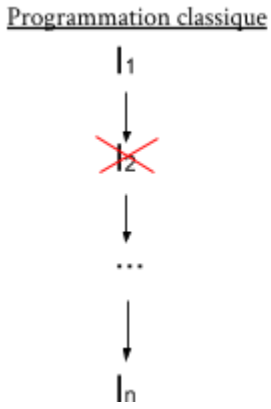
- Compréhension - Suivant l'organisation du développement, on comprend mieux qui fait quoi
- Réutilisation - Réutiliser l'architecture ou des éléments de l'architecture qui fonctionnent
- Construction - Guide sur la mise en place des éléments de base puis du reste
- Evolution - Si l'architecture est conçue pour évoluer, on peut facilement y ajouter des modules pour enrichir le système
- Analyse - Permet de voir si ce qu'on construit est conforme et cohérent
- Gestion - Pilotage des éléments du système

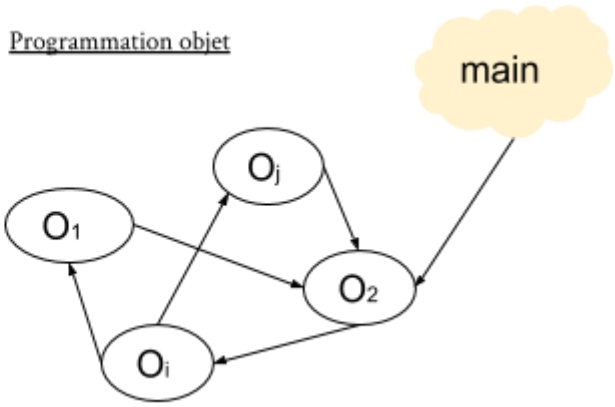
Qu'est-ce qu'un style architectural ?

Un style architectural définit une famille de systèmes logiciels ayant un vocabulaire commun pour désigner les composants, les caractéristiques topologiques (structure) et comportementales communes, et un ensemble de contraintes sur les interactions parmi les composants.

Exemples de styles courants : système clients-serveur (comportement commun), système orienté objet (vocabulaire commun : classes, objets, héritage | comportement commun: une sous-classe hérite d'une classe), système orienté service (tout est basé sur la notion de service), base de données (on stocke/récupère les données).

2. Style architectural objet



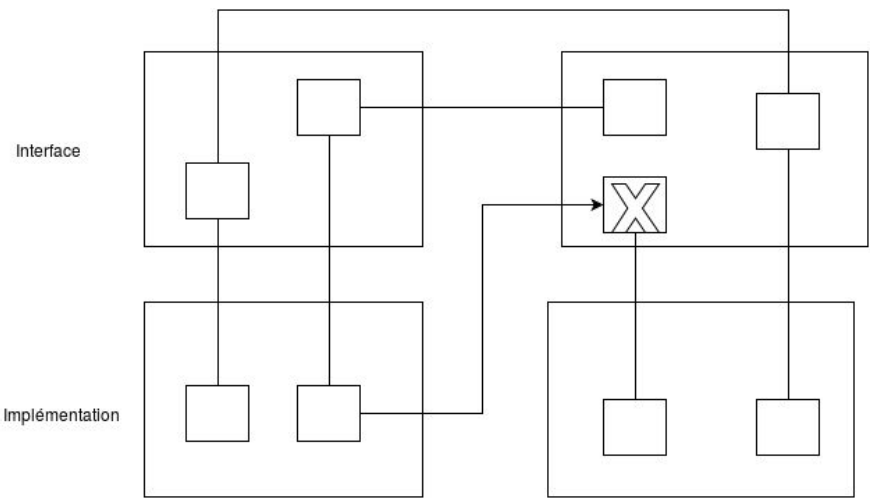


	Programmation classique	POO
Suppression de I2	On recompile tout	Poursuite sans l'objet

2.1 Limites SAO

Aujourd'hui les langages orientés objet sont les plus utilisés, mais il faut garder une vision critique sur ce type de langage.

- Granularité : Infiniment petit ou grand, permet de donner une "taille" à un objet (lorsqu'on est au niveau objet, on parle de petite granularité)
- Passage à l'échelle : confrontation à une quantité de données importante
- Fort couplage des objets (forte dépendance)
- La structure des applications objet est peu lisible
- La plupart des mécanismes objets (héritage, instanciation, polymorphisme, etc..) sont gérés manuellement
- Dans une architecture objet, on spécifie seulement les services fournis (méthodes) et on ne définit **en aucun cas** les besoins requis (pré-requis pour faire fonctionner les objets)
- Un objet ne fournit pas ou peu de support pour analyser les propriétés non fonctionnelles (sécurité, performance, etc…) car ces propriétés doivent être transversales
- Un objet prend difficilement en compte les évolutions (ajout, suppression, modification), on ne peut pas supprimer des attributs dans une classe par exemple
- Les modèles objets permettent difficilement la définition de l'architecture globale d'un système avant l'implémentation complète de ses composants
- *Cf schéma* - Il y a un couplage fort dans ce schéma
- Le schéma est considéré comme une architecture, sauf que celle-ci a trop de détails donc ce n'en n'est pas une
- Si l'interface X est supprimée, comment les implémentations auront accès à cette méthode ? Il faut regarder tous les objets s'ils font appel à cette méthode et mettre à jour le code



"Connaissance = information + raisonnement.

Information = le brut."

3. Style C&C (Composants et Connecteurs)

L'objet est au composant ce qu'est l'atome à la molécule. Un composant peut être un objet, mais le contraire n'est pas valable. "Un composant se doit d'apporter une fonctionnalité."

3.1 Définitions

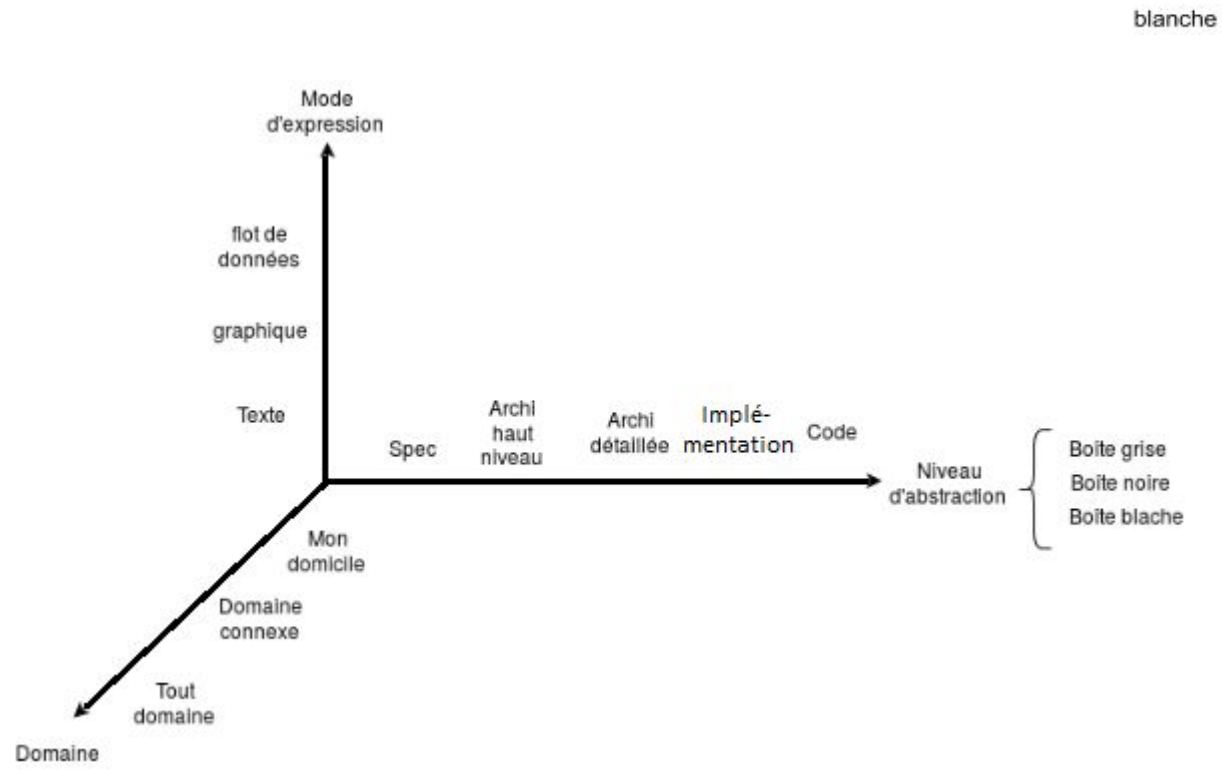
Un composant est une unité de composition qui spécifie une ou plusieurs fonctionnalités :

- Les composants fournissent des services et requièrent des besoins.
- Ils peuvent être déployés indépendamment et peuvent être composés avec d'autres composants.
- Ils sont préfabriqués, pré testés, et s'auto contiennent.
- Ils sont configurables et disposent de mécanismes d'introspection qui leur permettent de connaître et de modifier leurs caractéristiques.

Exemples de composants :

- ❖ Systèmes d'exploitation : Ré-utilisés par plusieurs personnes - composant à forte granularité
- ❖ Navigateur web
- ❖ Smartphones, toutes les applications (agenda…)

3.2 Les 3 dimensions d'un composant



Le niveau d'abstraction : on choisit le "niveau" de description du composant

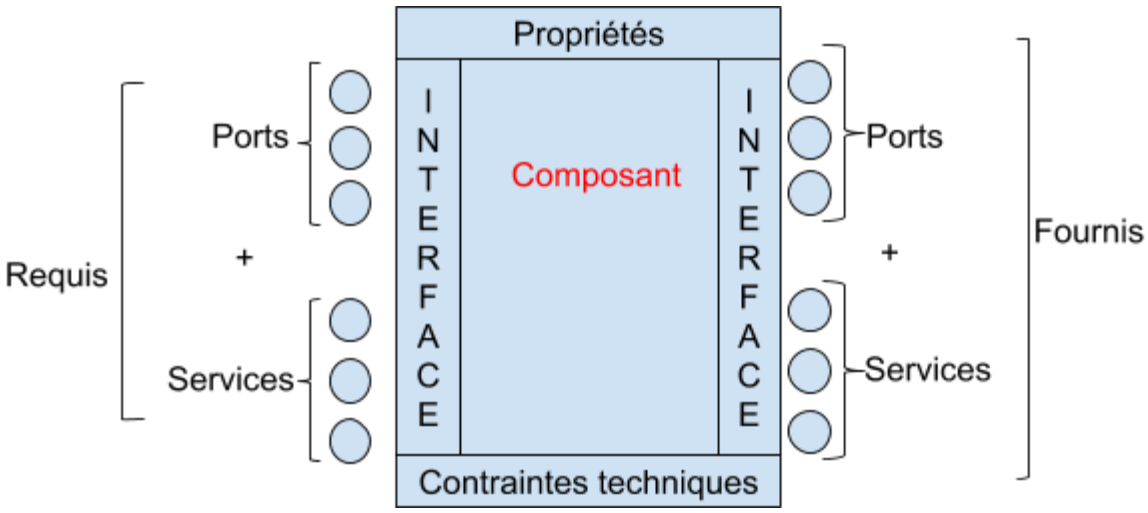
On donne un triplet pour définir un composant {Domaine, Mode d'expression, Niveau d'abstraction}

- Boîte noire : on ne sait pas ce qu'il y a dedans, on a seulement les entrées/sorties
- Boîte blanche : contraire de la boîte noire, on sait ce qu'il y a dedans, on peut utiliser et modifier ce qu'il y a dans la boîte blanche

- Boîte grise : on peut voir, on peut comprendre, mais on ne peut rien modifier, contrairement à la boîte blanche

L'objet est-il une boîte noire, grise ou blanche ?
- *Si on veut la boîte noire, on met les attributs et méthodes en private*
- *Boîte blanche : Tout en public*
- *Boîte grise : Accès seulement aux familles*

3.3 Représentation d'un composant



Un composant doit disposer d'interfaces :
- fournies
- requises

Une interface est composée de ports (permettent de dialoguer avec d'autres composants // avec les attributs des objets) et de services (jouent le rôle des méthodes). On a des ports et des services requis et des fournis. Un composant existe que s'il a un port et un service **fourni**.

Propriétés : Par exemple, qui a créé le composant. Elles sont propres au composant.
Contraintes techniques : Contraintes sur lesquelles on installe le composant (informations relatives à l'environnement d'installation du composant).

Exemples monde industriel :
- .NET → Programmation CCM (Comfort Control Module)
- EJB (Enterprise JavaBean)

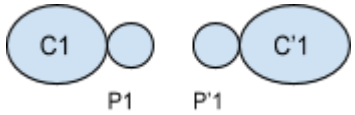
Exemples académiques :
- ACME (MIT)
- SOFA
- UNICOM

Connecteurs ou interactions

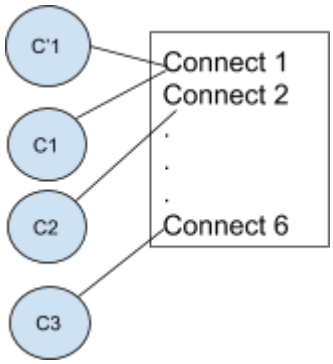
Connecteurs permettent de relier plusieurs composants entre eux. Trois types de connecteurs :

- **Implicites** : on prend deux composants avec un seul port (un port fournis : P1 sur un composant ; et sur l'autre un port requis : P'1). Connexion entre P1 et P'1 . Implicite car il n'a pas de nom.

Par exemple, en Java, un composant serait une classe Java, : C1, C2 : (ce qui est public est fourni), (ports requis avec des champs obligatoires), (accesseurs : port fournis), on peut relier les C1 et les C2 avec des tableaux, C2 en attribut de C1 (mais fort couplage) ….

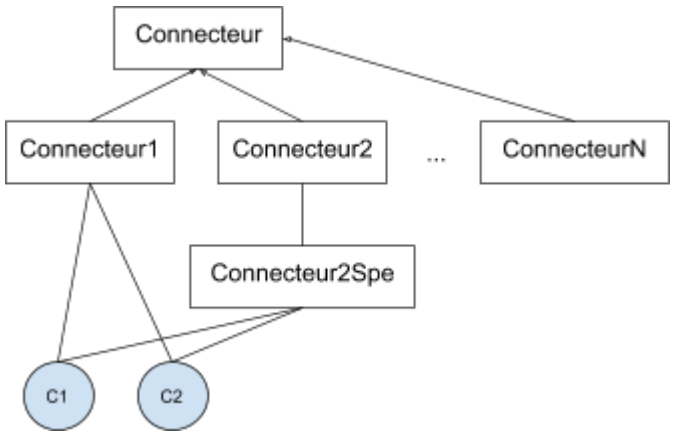


- **Énumérés** : on dispose d'une bibliothèque de connecteurs typés et énumérés et donc instanciés(Connect1, Connect2, …, Connect6). On dit ainsi que C1 et C'1 sont connectés avec Connect1.



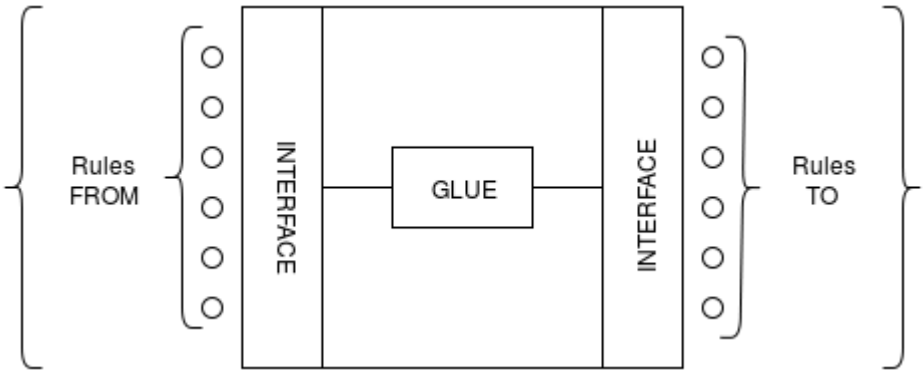
- **Explicites** : on dispose d'une classe de connecteurs, qu'on spécialise (Connect1, Connect2, … Connecti). Le connecteur devient presque au même niveau que le composant, ce n'est pas juste un lien, il a sa propre sémantique.

En Java, on pourrait considérer les connecteurs explicites comme des classes.



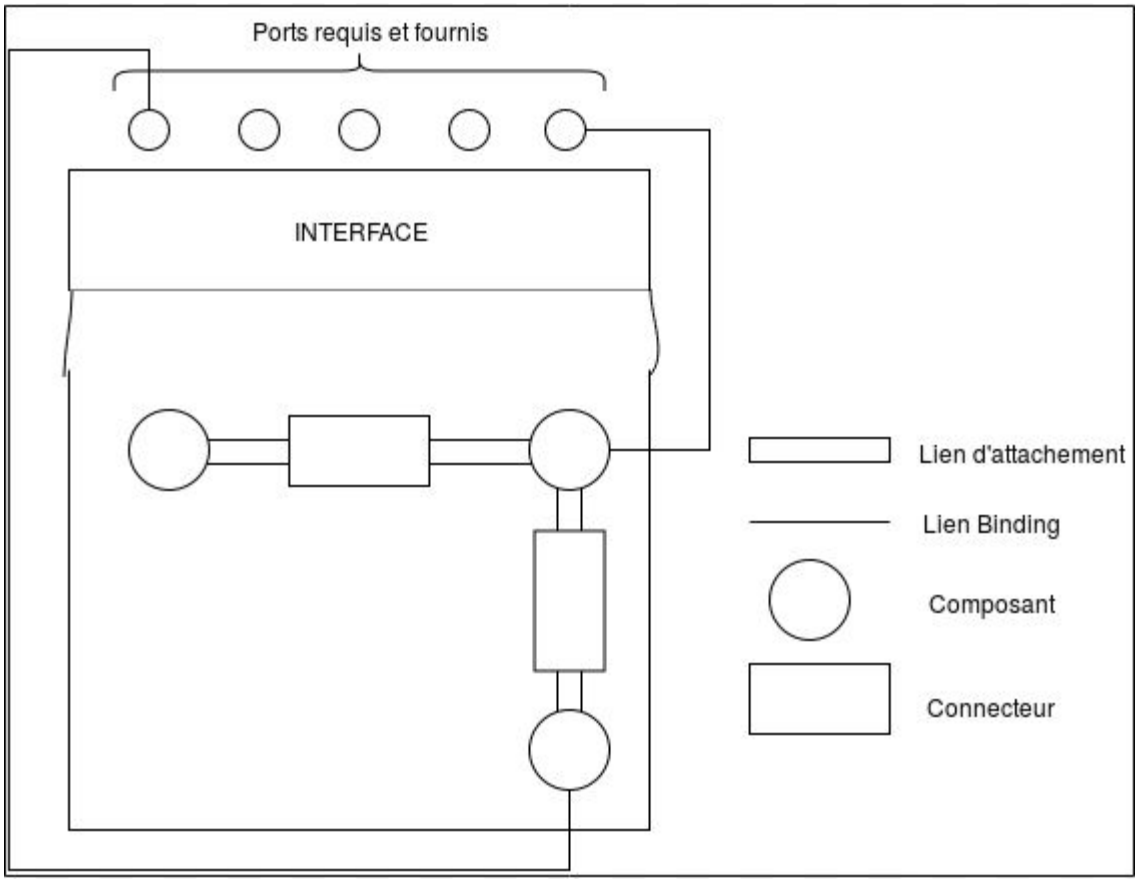
Représentation d'un connecteur

Port pour composant = rôle pour connecteur . Il peut y avoir une ou plusieurs glues suivant les ensembles en entrée et en sortie.



3.4 Configuration

Configuration : Graphes de composants et de connecteurs

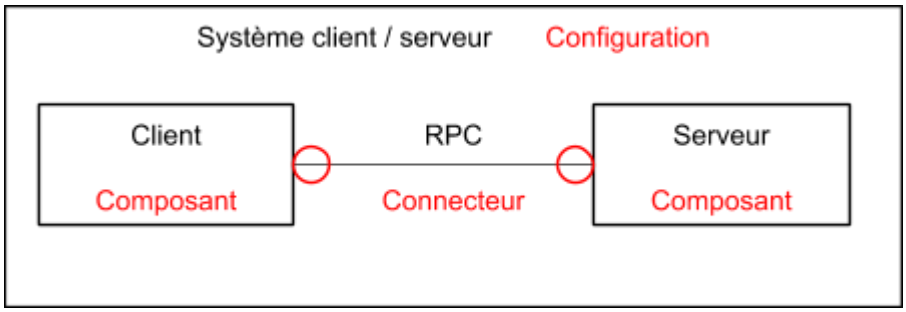


Attachement :

- Interaction entre le port fourni d'un composant et le rôle requis d'un connecteur.
- Interaction entre le port requis d'un composant et le rôle fourni d'un connecteur

Binding : l'information qui rentre dans le port (on protège les composants ils ne peuvent pas communiquer directement), l'interface de la configuration transmet ensuite l'information reçu sur le port au bon composant.

Exemple client/serveur (RPC = Remote Protocol Communication)

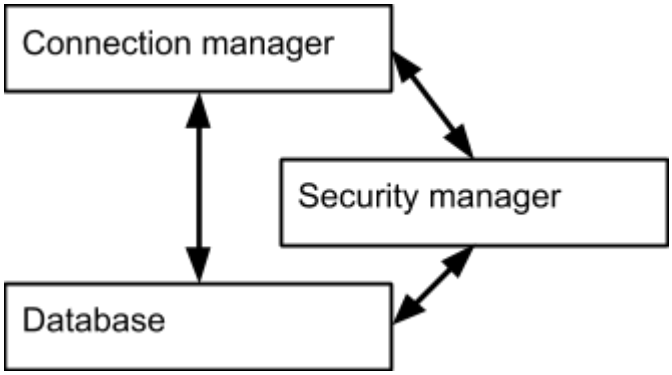


Client / Serveur en langage ACME : https://www.cs.cmu.edu/~acme/docs/language_overview.html

```
System SimpleCS = {
        Component Client = { Port Send_Request }
        Component Server = { Port Receive_Request }
        Connector RPC = { Roles { Caller, Called } }

        Attachements : {
                Client.SendRequest to RPC.caller,
                Server.Receive_Request to RPC.called
        }
}
```

Zoom sur le serveur : (serveur détail)



Connection_Manager :

- External_Socket : il est extérieur et est relié avec le RPC
- Security_Check : entre le connecteur manager et le security manager
- DB_Query : entre le connection manager et la database

Security Manager :

- Security_Autentification : connection entre le connection manager et le security manager
- Check_Query : connection vers la database

Database :

- Security.Management
- Query_Interrogation

Note : Le code suivant remplace la ligne en rouge sur le code précédemment présenté.

```
Component Server = {  Port Receiver ReceiverRequest,
                      System Server_Detail =
                            {
                                  Composant ConnectionManager =
                                              { Port External Socket }
                                              { Port Security Check }
                                              { Port DB_Query }
                            }
                      }
```

On a au total 6 attachments entre les différents composants et un binding avec le external socket et le RPC.

Projet A.S.A.

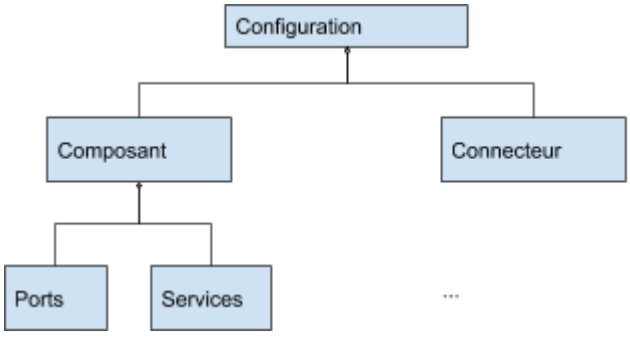
Pré-requis :

- programmation JAVA
- UML

OMG	POO			ASA
M2	Class (concept - définition des concepts du paradigme, ici prog objet)	Diagramme class_uml class instance / sous-classe	métamodèle	Component, connector, configuration, binding, etc :)
M1	Personne (classe)		modèle	Système client - serveur
M0	Toto (instanciation)		Application, instance	Instance du système client - serveur

Métamodèle objet : établissement des relations entres les éléments

1. Conception et spécification de A.S.A : Etablir le diagramme de classes du niveau M2



Evaluation du projet :
7 points sur la spécification / conception
13 sur le codage

+ document expliquant en quoi consiste ASA (intro), spécification en les expliquant , et expliquer les choix dans la programmation et de ce change par rapport à la spécification

13 avril