

Architecture et Styles d'Architectures

<https://github.com/lasherme/AsaPerezLasherme.git>

Introduction

Au cours de cette unité d'enseignement, nous avons pu observer plusieurs styles d'architectures, et plus particulièrement comment fonctionne une architecture par composant. De ce fait, dans notre projet, nous avons dû réaliser une architecture de A à Z. En premier lieu, nous avons dû concevoir cette architecture par composant vu en cours. Une fois cela réalisé, nous avons dû adapter le modèle bien connu du client-serveur à notre architecture par composant préalablement créée. Et enfin nous avons instancier le modèle client-serveur réalisé grâce à l'architecture par composant conçu.

1^{ère} partie : Conception

M2

Pour commencer la metamodélisation du projet, nous devons créer un style d'architecture par composant. Comme nous pouvons le voir sur le [metamodèle](#), l'architecture doit commencer par une configuration, qui elle-même peut contenir d'autres sous-configuration. Une configuration est composée d'interfaces afin de pouvoir communiquer avec d'autres composants extérieurs. De plus, une configuration possède des composants et des connecteurs. Un composant peut soit être primitif, soit composite. Un composant primitif, est un composant possédant une interface fournie et d'une autre requise dans lesquelles nous pouvons y retrouver des ports. Un composant composite n'est rien d'autre qu'un composant primitif dans lequel nous pouvons y stocker d'autres composants ou connecteurs. Les connecteurs sont comme les composants, soit primitif soit composite. Les connecteurs permettent de relier les composants entre eux à l'aide de lien d'attachement. De plus, ils permettent un tas de fonctionnalités diverses grâce à la `glue` qui les compose. Cette `glue` peut permettre par exemple de traduire les informations d'un composant A vers un composant B, servir de sécurité, etc... Les composants possèdent deux interfaces, une `FROM` et une autre `TO` — équivalentes aux interfaces fournies et requises. Sur ces interfaces, nous pouvons y retrouver des rôles — équivalents aux ports des composants. Pour relier des composants entre eux nous créons un lien d'attachement entre le composant A vers le connecteur, et un autre lien d'attachement du connecteur vers un composant B. En fait, nous connectons un port requis du composant à un rôle fourni du connecteur et vis-versa. Et pour finir, si nous voulons relier un port d'un composant à un composant extérieur de la configuration, nous établissons un lien binding entre le port concerné et un port de sa configuration afin qu'un port externe puisse avoir accès aux fonctionnalités du composant interne de la configuration.

M1

Dans cette partie, nous devons donc créer un [modèle](#) client-serveur basé sur l'architecture à composant préalablement conçu. Nous avons donc créé une configuration client-serveur générale qui englobe tout le reste. Nous avons donc créé un composant client et un composite composant serveur. Ceux-ci sont reliés grâce à un connecteur `RPC`. Par le biais de ce connecteur, le client peut communiquer au serveur grâce à la méthode `sendRequest` et le serveur peut écouter ses clients grâce à la méthode `receiveRequest`. A l'intérieur du composite composant serveur, nous pouvons y retrouver 3 autres sous-composants : `database`, `connectionManager`, `securityManager`. Ces composants sont reliés les uns aux autres à l'aide de connecteurs. Cela permet ainsi de les faire communiquer entre eux à l'aide de leurs ports. Les connecteurs permettent de faire la liaison entre chaque composants. Ainsi ils permettent de traduire ou même de sérialiser l'information. Pour finir, `connectionManager` a besoin de pouvoir communiquer avec l'extérieur, c'est pour cela qu'il relie un de ses ports à un port de la configuration à l'aide d'un lien binding.

2^{eme} partie : Instanciation

```

import composant.*;
import compositeComposant.CompositeComposantServer;
import connecteur.*;
import configuration.*;
import glue.*;
import intf.*;
import lienAttachement.*;
import lienBinding.LienBindingExternalSocketComposantExternalSocketConfiguration;
import port.*;
import service.*;

public class Main {
    public static void main(String args[]) {

        // Port
        InterfacePort PortRequisClient = new Port("portRequisComposantClient");

        // Glue
        InterfaceGlue GlueAuthentification = new GlueAuthentification("authentification");
        InterfaceGlue GlueDatabaseConnection = new GlueDatabaseConnection("dataBaseConnection");
        InterfaceGlue GlueSerialiseCommunication =
            new GlueSerialiseCommunication("serialise_and_communication");
        InterfaceGlue Glue = new Glue("glueConnecteur");

        // Service
        InterfaceService ServiceRequis = new Service("serviceRequis");
        InterfaceService ServiceFourni = new Service("serviceFourni");

        // Composant
        InterfacePrimitiveComposant ComposantClient =
            new PrimitiveComposantClient(ServiceRequis, ServiceFourni, PortRequisClient);
        InterfacePrimitiveComposant ComposantDataBase = new PrimitiveComposantDataBase();
        InterfacePrimitiveComposant ComposantConnectionManager =
            new PrimitiveComposantConnectionManager();
        InterfacePrimitiveComposant ComposantSecurityManager = new PrimitiveComposantSecurityManager();

        // Connecteur
        InterfaceConnecteur ConnecteurAuthentification =
            new ConnecteurAuthentification("connecteurAuthentification", GlueAuthentification);
        InterfaceConnecteur ConnecteurDataBaseConnection =
            new ConnecteurDataBaseConnection("connecteurDataBaseConnection", GlueDatabaseConnection);
        InterfaceConnecteur ConnecteurRPC = new ConnecteurRPC(GlueSerialiseCommunication);
        InterfaceConnecteur Connecteur = new Connecteur("connecteurCompositeComposant", Glue);

        // Configuration
        InterfaceConfiguration ConfigurationServeur = new ConfigurationServeur();

        // Composite Composant
        CompositeComposantServer compositeComposantServer =
            new CompositeComposantServer(Connecteur, ConnecteurDataBaseConnection, ConfigurationServeur);

        // Lien Attachement
        LienAttachementRoleCalledPortDbQuery lienAttachementRoleCalledPortDbQuery =
            new LienAttachementRoleCalledPortDbQuery(
                Connecteur.getRoleFourni(), ComposantConnectionManager.getPortRequis());
        LienAttachementRoleCalledPortReceiveRequest lienAttachementRoleCalledPortReceiveRequest =
            new LienAttachementRoleCalledPortReceiveRequest(
                ConnecteurRPC.getRoleFourni(), ConfigurationServeur.getPortRequis());
        LienAttachementRoleCalledPortSecurityCheck lienAttachementRoleCalledPortSecurityCheck =
            new LienAttachementRoleCalledPortSecurityCheck(
                ConnecteurAuthentification.getRoleFourni(), ComposantConnectionManager.getPortRequis());
        LienAttachementRoleCalledPortSecurityManagement lienAttachementRoleCalledPortSecurityManagement =
            new LienAttachementRoleCalledPortSecurityManagement(
                ConnecteurDataBaseConnection.getRoleFourni(), ComposantDataBase.getPortRequis());
        LienAttachementRoleCallerPortCheckQuery lienAttachementRoleCallerPortCheckQuery =
            new LienAttachementRoleCallerPortCheckQuery(ConnecteurDataBaseConnection.getRoleRequis(),
                ComposantSecurityManager.getPortFourni());
        LienAttachementRoleCallerPortQueryInterrogation lienAttachementRoleCallerPortQueryInterrogation =
            new LienAttachementRoleCallerPortQueryInterrogation(
                Connecteur.getRoleRequis(), ComposantDataBase.getPortFourni());
        LienAttachementRoleCallerPortSecurityAuthentification
        LienAttachementRoleCallerPortSecurityAuthentification =
            new LienAttachementRoleCallerPortSecurityAuthentification(

```

```

    ConnecteurAuthentication.getRoleRequis(), ComposantSecurityManager.getPortFourni());
LienAttachementRoleCallerPortSendRequest LienAttachementRoleCallerPortSendRequest =
    new LienAttachementRoleCallerPortSendRequest(
        ConnecteurRPC.getRoleRequis(), ComposantClient.getPortFourni());

// Lien Binding
LienBindingExternalSocketComposantExternalSocketConfiguration
LienBindingExternalSocketComposantExternalSocketConfiguration =
    new LienBindingExternalSocketComposantExternalSocketConfiguration(
        compositeComposantServer.getConfigurationServeur().getPortFournis(),
        ComposantConnectionManager.getPortFourni());
    }
}

```

Conclusion

Pour conclure nous avons beaucoup appris au cours de ce projet. Tout d'abord la création du M2 a nécessité beaucoup de recherches afin de mieux comprendre ce qui était attendu. Dans ce premier diagramme nous avons ainsi pu définir le méta Modèle de l'architecture du projet, c'est ici que les interfaces étaient définies, c'est ainsi que nous avons défini notre modèle client/serveur. Vient ensuite le M1, le diagramme de classe qui instancie les éléments du M2. Dans ce dernier nous avons liés les différents composants entre eux afin de rendre fonctionnelle et cohérente notre architecture. Le point qui nous a posé le plus de soucis fut les **glues**. En effet nous ne comprenions pas au premier abord l'utilité de cet élément, c'est en cherchant dans vos publications que nous sommes tombés sur l'article "Metamodel for Architecture Description Language Based on First-Order Connector Types" qui, nous a facilité la compréhension du projet. Enfin nous avons procédé à l'implémentation des diagramme en Java, nous aurions pu procéder avec le langage ACME, cependant à cause de la crise sanitaire nous n'avons pas pu faire le projet dans sa globalité. Nous avons préféré utiliser un langage que nous connaissions déjà.