HADL

Chénais Sébastien – Wollenburger Antoine

Table des matières

[Introduction 2](#_Toc374175802)

[Niveau M2 2](#_Toc374175803)

[Description d’HADL 2](#_Toc374175804)

[Diagramme d’HADL 5](#_Toc374175805)

[Diagramme global 5](#_Toc374175806)

[Explication du diagramme 5](#_Toc374175807)

[Niveau M1 6](#_Toc374175808)

[Première approche 6](#_Toc374175809)

[Seconde approche 6](#_Toc374175810)

[Niveau M0 8](#_Toc374175811)

[M2 8](#_Toc374175812)

[M1 8](#_Toc374175813)

[Le M0 10](#_Toc374175814)

[Gestion de l’instanciation 10](#_Toc374175815)

[Gestion de la communication 10](#_Toc374175816)

[Conclusion 10](#_Toc374175817)

# Introduction

Le but de ce projet est de construire un langage de description d’architecture maison : HADL. L’application se fera sur un modèle de composant client-serveur, avec un serveur contenant un gestionnaire de connexions, un gestionnaire de base de données et un gestionnaire de sécurité.

Dans un premier temps, il vous sera décrit le méta modèle de niveau M2 (architecture d’infrastructure) puis son implémentation au niveau M1 (architecture d’application). L’étape M0 ne sera pas décrite ici, mais est déduite du M1.

# Niveau M2

Ce niveau permet de définir le langage architectural, ses mécanismes et ses composants. Nous avons utilisé UML pour décrire HADL.

## Description d’HADL

Les éléments nécessaires à ce langage sont :

* Composant
  + Port/service
* Connecteur
  + Glue
  + Rôle
* Configuration
  + Port/service
* Lien
  + Binding
  + attachment

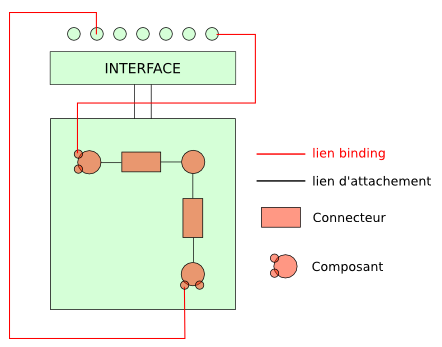


Figure 1- représentation des composants d'HADL

HADL est aussi fourni avec les mécanismes suivants :

* composition : le fait qu’un composant en utilise un autre
* héritage : le fait qu’un composant prenne les caractéristiques d’un autre
* raffinement : le fait de pouvoir faire une mise au point sur un composant pour avoir plus d’information

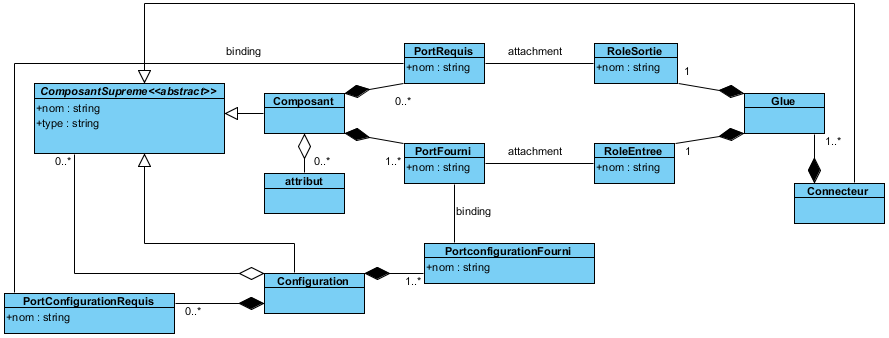


Figure 2 - Diagramme UML d'HADL

## Diagramme d’HADL

### Diagramme global

Cf. figure 2

### Explication du diagramme

Les différents éléments d’HADL sont représentés ici : le composant avec ses ports/services, le connecteur avec sa glue et ses rôles et la configuration avec ses ports/services.

#### La configuration

Une configuration peut être composée de connecteurs, de composants et de configurations. Ces composites sont représentés par la classe abstraite ComposantSupreme et l’agrégation faible entre la configuration et cette classe abstraite.

Les ports d’une configuration ne peuvent être liés qu’au port du même type d’un composant. C’est-à-dire qu’un port fourni d’une configuration ne pourra être lié qu’à un port fourni d’un composant. Le lien sera alors un lien dit de « binding ». Une configuration possède au moins un port fourni.

Du fait de la présence d’au moins un port fourni, la configuration possède au moins un composant.

#### Le composant

Le composant hérite de ComposantSupreme. Il possède au moins un port fourni et peut posséder des ports requis. Un port fourni peut être lié à un rôle en entrée d’un connecteur par un lien dit « d’attachment » ou à un port fourni de configuration comme expliqué précédemment. Un port requis peut être lié à un rôle de sortie d’un connecteur ou à un port requis de configuration.

Un composant peut être composé d’attributs.

#### Le connecteur

Le connecteur est composé d’au moins une glue. Chaque glue est composée d’un rôle en entrée et d’un en sortie. Les rôles peuvent se lier aux ports des configurations comme expliqué précédemment.

# Niveau M1

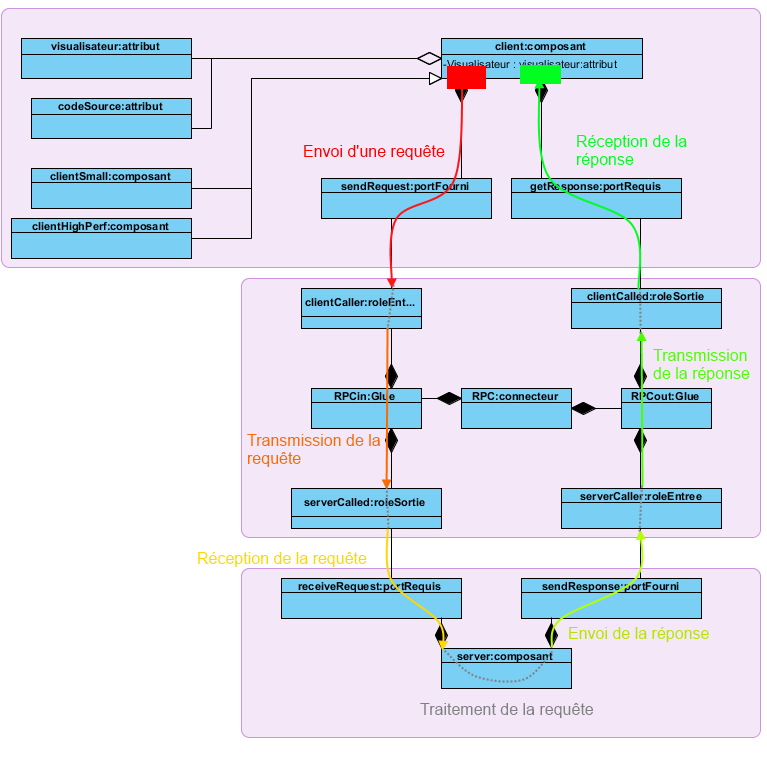
Première approche

Figure 3 - M1 client-serveur sans raffinement

Dans cette première approche, nous avons modélisé le serveur comme un composant unique.

L’envoi d’un message au serveur par le client se déroule comme suit : le client émet un message, message relayé par le connecteur RPC au serveur. Ce message est ensuite traité par le serveur, qui envoie sa réponse au client via le connecteur RPC.

## Seconde approche

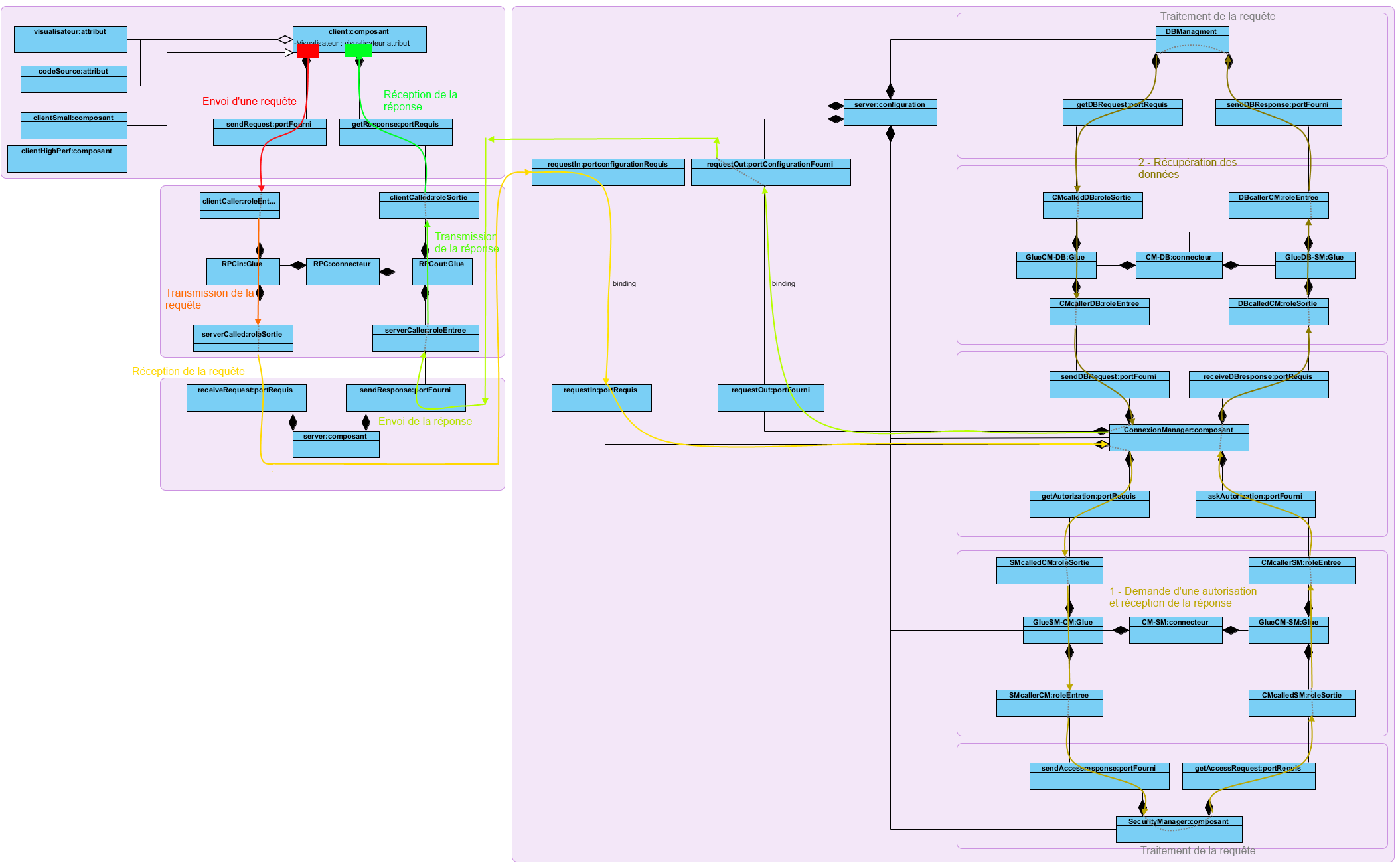
****

Figure 4 - M1 client-serveur raffiné

Le serveur est ici considéré comme une configuration, et lié au premier modèle via des liens de binding.

Le traitement interne au serveur du message se déroule en deux étapes :

* Tout d’abord, le connexion manager, qui a reçu le message par les liens de binding, demande une autorisation au security manager, via le connecteur cm-sm.
* Cette autorisation reçue, le connexion manager demande la récupération de données au DBmanager, via le connecteur cm-db.

Ensuite, les données récupérées par le connexion manager sont renvoyées au client, via les liens de binding et le connecteur RPC.

# Niveau M0

Le développement en java du M0 correspond à une instanciation du M1. Et le M1 est censé être une instanciation du M2. En java cependant, cette double instanciation n’est pas permise. On utilise donc le mécanisme d’héritage entre le M2 et le M1 afin de pouvoir instancier le M0.

## M2

La première étape consiste à créer le M2. Il n’y a aucune difficulté à ce niveau, il suffit de transcrire le schéma. Par anticipation, on se rend compte que la communication entre composants, connecteurs et configurations va être plus complexe que prévu.

## M1

Le M1 est donc obtenu en sous classant le M2. Le nombre de classe augmente rapidement. Par exemple, nous avons besoin d’une classe « Client » et d’une classe « Serveur » qui hérite de Composant. Chacune de ces classes possède au moins un port en entrée et un port en sortie.

Nous remarquons que nous avons très peu d’information à propos de la configuration globale. Cela implique un traitement spécifique au lancement qui n’est pas acceptable. Nous ajoutons donc une configuration « ClientServeur » avec ses « portConfiguration ».

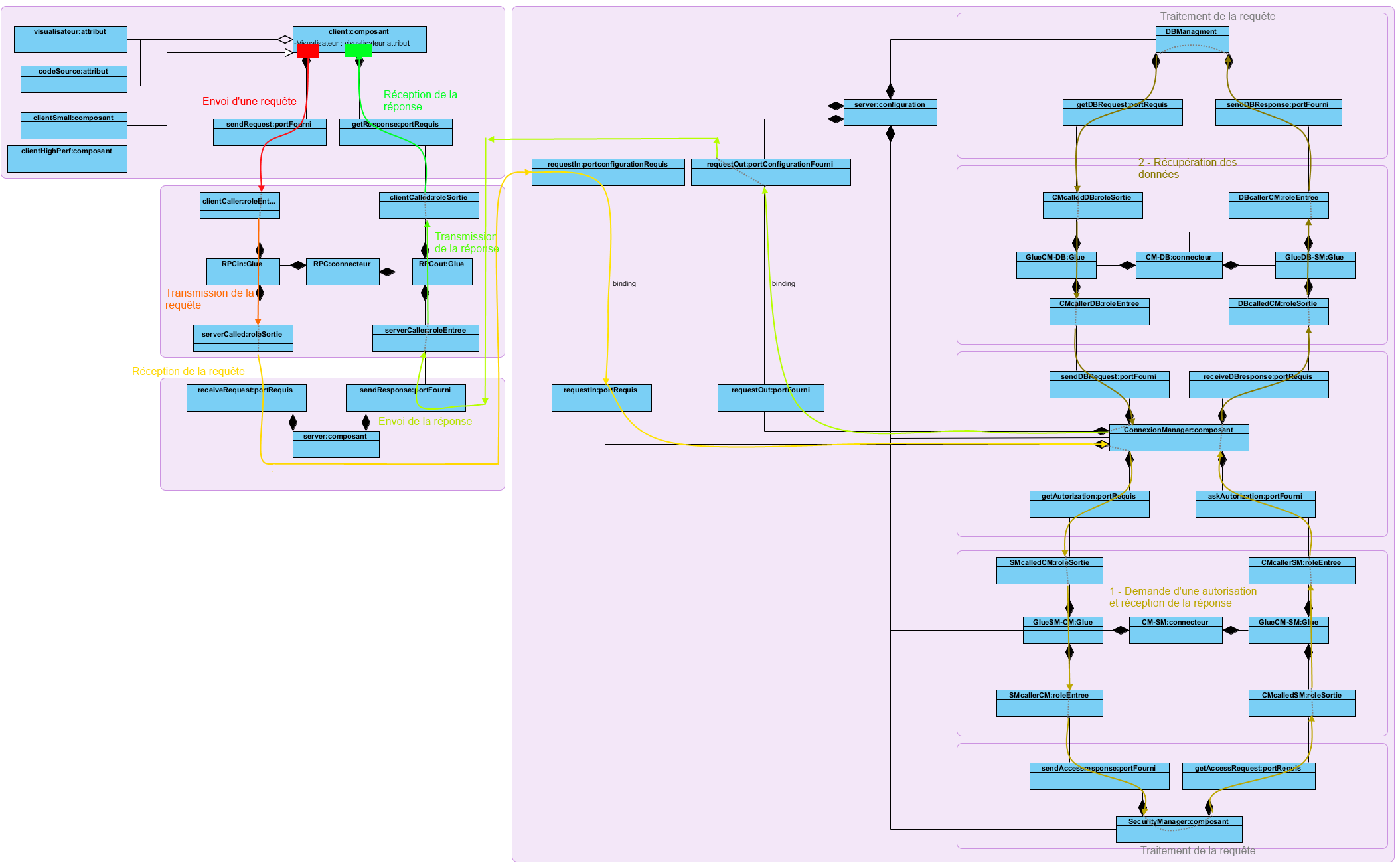
****

Figure 5 – M1 corrigé

## Le M0

Une fois le M1 créé, le M0 est plus simple à mettre en place. Nous créons une classe utilisatrice qui instancie dans l’ordre correct nos classes du M1. Chaque classe du M1 possède alors une et une seule instanciation.

### Gestion de l’instanciation

L’instanciation se fait en deux parties, correspondant aux deux configurations. Nous commençons par la configuration « ClientServeur », en considérant le serveur comme un composant simple. Ensuite, nous faisons la mise au point sur la configuration « Serveur ».

La première difficulté vient d’une erreur de modélisation : un port ou rôle ne connait pas son parent. Il est impossible de faire transiter des messages d’un port/rôle vers le composant/connecteur/configuration associé. Nous modifions alors le M2 pour résoudre ce problème. Un problème en découle alors, qui instancier en premier. Nous décidons d’instancier les ports/rôles en premier, et d’enregistrer leur parent au moment de leur ajout (constructeur ou méthode d’un composant/connecteur). Nous répétons cette méthode pour les glues et les ports configurations.

### Gestion de la communication

Une fois l’ensemble instancié, il faut les faire communiquer. De pars le fonctionnement d’un système à composant, les composants ne connaissent pas leurs voisins. Il faut donc systématiquement passer par la configuration pour la transmission de message.

Lorsqu’un message est transmis à la configuration, on lui passe en sus du message le nom de la classe. La configuration regarde donc qui est l’appelant, et consulte ses « bindings » et « attachements » pour déterminer le destinataire. La configuration possède donc une liste de règle pour les transmissions.

On remarque pourtant un comportement inattendu quand un port possède à la fois un « binding » vers un port configuration et un « attachement » vers un rôle. La configuration appelle les deux liens. En pratique, ce cas n’arrive pas dans notre système, les ports pour « bindings » étant spécifiques à ce but.

Du côté du destinataire, une méthode récupère le message et lance le traitement associé (transformation du message pour la glue, traitement pour le composant, …) avant de transmettre le message à son tour.

### Cas particuliers

Nous avons rencontré une difficulté au niveau du passage au niveau configuration serveur. En effet, dans notre façon de faire, nous passons d’un port « entrée » du composant  « serveur » vers le port configuration de la configuration « serveur », puis de ce dernier vers le port « entrée » du composant « connexionManager ». Ce n’est pas possible puisqu’un port « entrée » ne peut pas faire transiter des messages vers l’extérieur. Nous avons donc modifié le M1 en ajoutant deux ports dédiés à la mise à l’échelle pour le composant « serveur » et deux nouveaux ports configurations pour la configuration « serveur ».

### La communication

Le passage à l’échelle, c’est-à-dire l’utilisation de « serveur » comme composant ou configuration au niveau de l’utilisateur se détermine à l’exécution. En pratique, on détermine si le message est affiché ou non au niveau de la configuration. Dans le cas où « serveur » est vu comme un composant, la configuration serveur effectue son travail silencieusement. Du point de vue utilisateur, on aura un message en entrée de serveur et un message en sortie. Sinon, on aura le trajet complet à l’intérieur de la configuration.

# Conclusion

HADL permet de simplifier grandement la lisibilité d’un modèle sur une architecture complexe. De plus, l’architecture par composants permet la réutilisabilité ceux-ci.