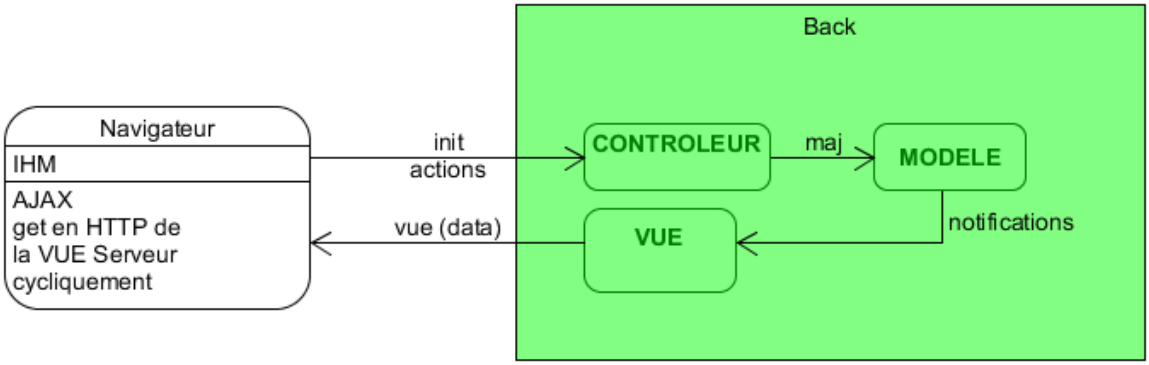
Exercice 02 : Correction

Un site de salon pour jouer au poker

La correction de l'exercice mettant en pratique la réalisation d'une architecture J2EE.

1. Introduction

Pour réaliser la gestion de la partie de jeu du SI (les autres parties seront fait classiquement), nous allons utiliser le Design Pattern d'Architecture **MVC** adapté à l'interaction entre un navigateur et le serveur . Le principe en est le suivant :



Le principe de base est d'utilisé le modèle MVC dont la VUE est un composant déployé sur la partie "back". Elle contient toutes les informations dynamiques qui doivent être affichées dans la navigateur. A la charge du navigateur de récupérer cycliquement ces informations.

Ce type d'architecture est à utiliser avec la technologie AJAX dont les requêtes http se font sans rechargement de la page (sauf si l'action entraine un changement de page (qui sera demandé par le navigateur)).

Au chargement de la page, la requête "init" permet de demander au modèle de notifier toutes les informations à la VUE.

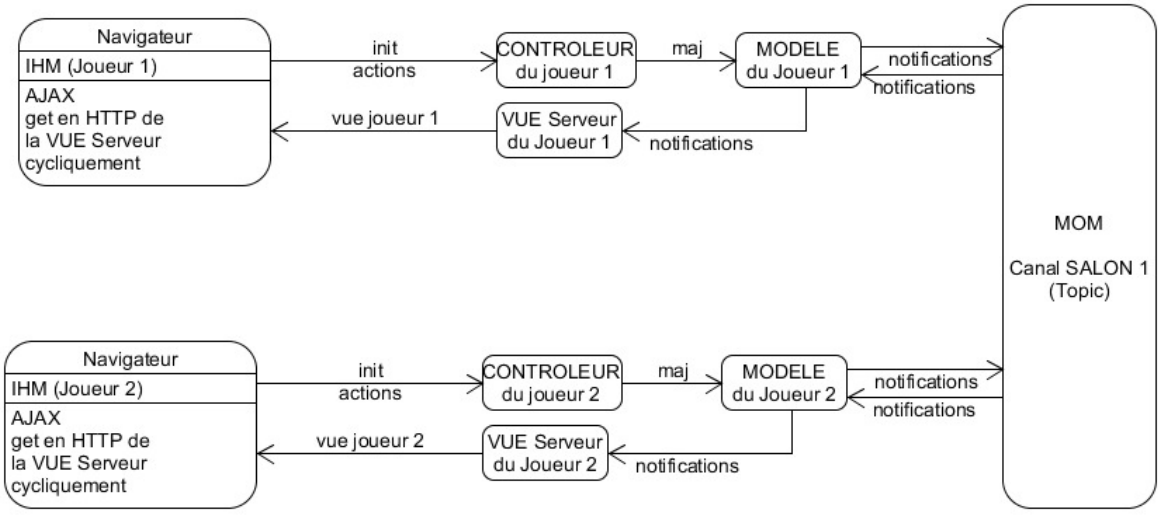
Une action réalisée par l'utilisateur sur le navigateur est envoyée au composant CONTROLEUR qui retourne au moins l'acceptation de la requête (il peut retourner un cas d'erreur, …).

Ensuite, le CONTROLEUR fait la mise à jour du MODELE qui à son tour notifie la VUE d'un changement d'état du MODELE.

Et comme le navigateur interroge cycliquement la VUE pour récupérer les informations, la page se met à jour.

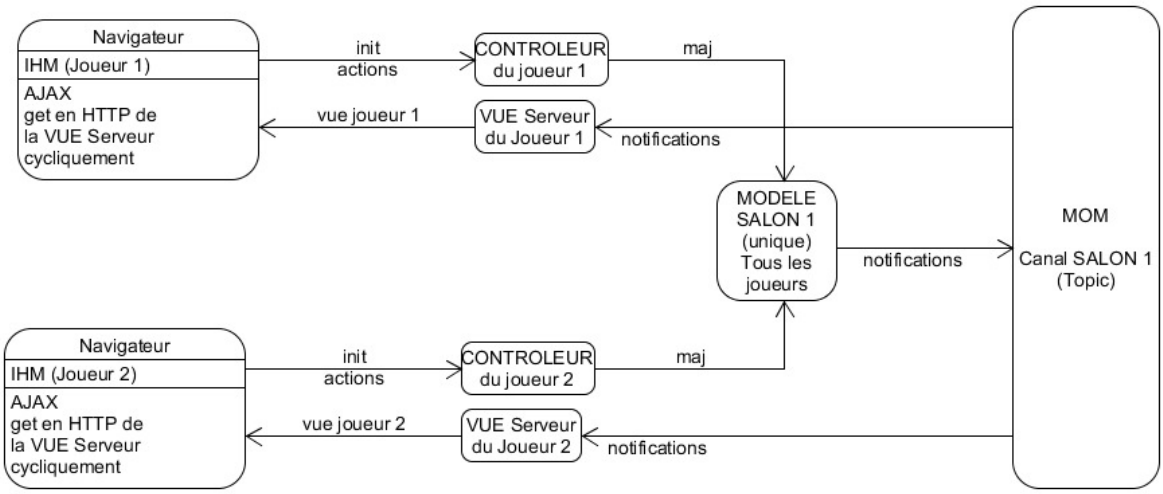
2. La configuration architecturale

Nous prenons ce principe pour réaliser notre architecture mais adapté à notre table de jeu dans la mesure où l'on va "synchroniser" la maj de chacun des MODELES des joueurs via l'utilisation d'un MOM. On obtient le principe précédent adapté suivant :



Ainsi, chaque action d'un joueur se répercute sur les MODELES des autres joueurs dont les VUES sont naturellement mises à jour , et donc la page qui représente la table de POKER de chacun des joueurs.

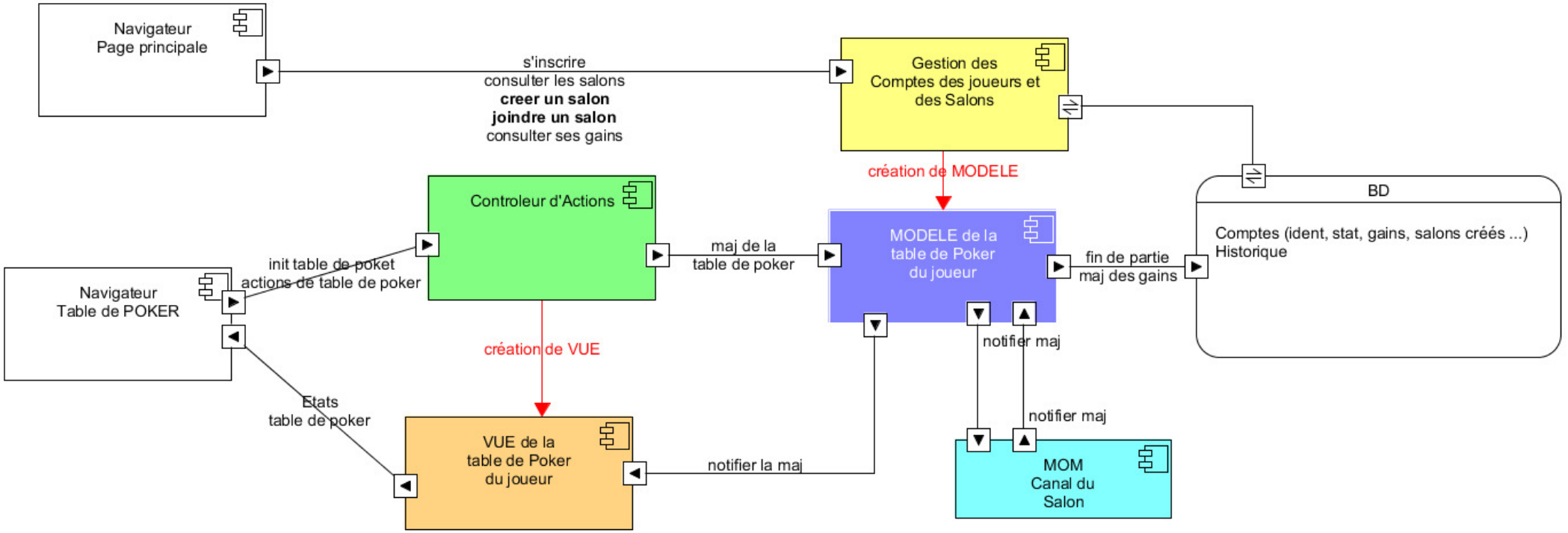
On peut aussi adopter un autre principe dans lequel le MODELE est unique pour tous les joueurs (d'un SALON donné) :



Dans ce cas, le MODELE est créé lors de la création du SALON.

Il est à noter que l'on peut simplifier en n'utilisant pas de MOM, dans la mesure où c'est le MODELE qui notifie lui-même toutes les VUES (plus facile à mettre en place si on n'a pas de MOM sous la main).

Si on prend le premier principe pour réaliser notre SI, on obtient la configuration architecturale suivante :

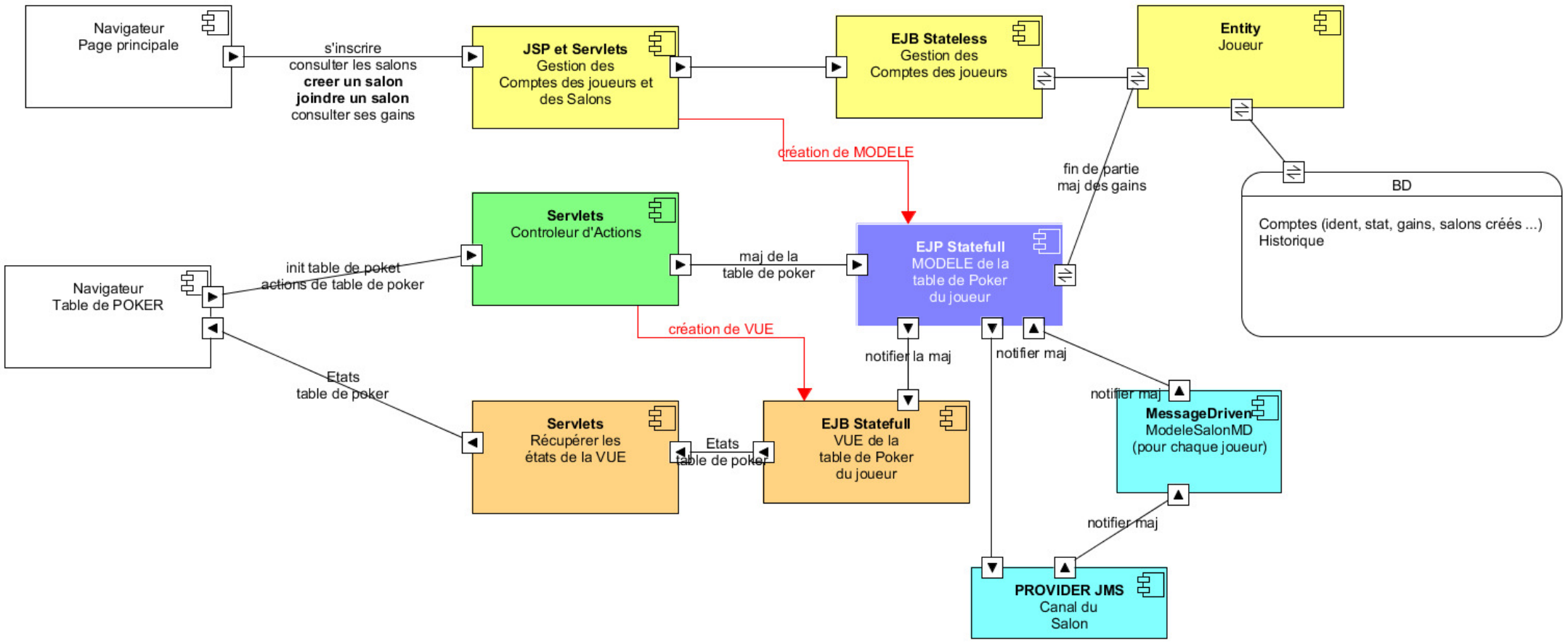

Navigateur
Page principale
s'inscrire
consulter les salons
creer un salon
joindre un salon
consulter ses gains
Gestion des
Comptes des joueurs et
des Salons
création de MODELE
BD
Comptes (ident, stat, gains, salons créés ...)
Historique
Controleur d'Actions
init table de poket
actions de table de poker
maj de la
table de poker
MODELE de la
table de Poker
du joueur
fin de partie
maj des gains
Navigateur
Table de POKER
création de VUE
notifier maj
Etats
table de poker
notifier maj
VUE de la
table de Poker
du joueur
notifier la maj
MOM
Canal du
Salon

Commentaires :

- Sur la page principale, toutes les actions hors celles réalisées pour jouer dans un salon sont traitées par la **Gestion des Comptes des joueurs et des Salons** (inscription, création d'un salon (type de jeu, configuration, …), joindre un salon, consultation des salons existants (créés mais fermés, créés et ouverts), consultation de ses gains, ….
- Quand un joueur joue, il utilise la page de Table de POKER à partir duquel le joueur réalise ses actions qui sont traitées le **Contrôleur d'Actions**.
- Le propriétaire (quand il ouvre le salon) et les autres joueurs (quand ils rejoignent le salon) ont chacun un **Modèle de la table de Poker** qui contient un état de la table de poker. Chacun de ces modèles est abonné au canal du salon auquel il appartient.
- Quand un joueur réalise une action dans son navigateur, en fonction de l'action reçue son modèle se met à jour :
  - le modèle réalise un traitement en fonction de l'action reçue, met à jour son modèle et notifie les changements à tous les autres modèles.
  - chacun des modèles notifie sa vue. Le traitement réalisé dans chaque vue se fait en fonction de l'appartenance du joueur.
- Le navigateur de chacun des joueurs récupère cycliquement le nouvel état de sa vue et met à jour le contenu de l'IHM en conséquence.

3. L'Architecture technique

Le déploiement de la Configuration Architecturale dans une architecture Technique de type J2EE est la suivante :



ARCHITECTURE TECHNIQUE DU JEU DE POKER

En utilisant la technologie J2EE, nous faisons les choix suivants :

- Le **ControleurActions** est composé de Servlets qui prennent en charge les actions de la table de Poker.
- Le **Gestion des Comptes des joueurs et des Salons** est l'enchainement des composants de JSP et Servlets et d'un EJB Stateless qui enregistre, met à jour, consulte les Comptes des joueurs et les Salons. Les Comptes et Salons sont des Entity afin d'utiliser la persistance pour la mise à jour en BD.
- Le EJB Stateful **Modele table de Poker** contient l'algorithme de gestion de la table de POKER d'un joueur donné.

  On ne rentrera pas dans le détail de la dynamique du jeu mais il nous faux montrer comment se fait le mode opératoire d'initialisation et la collaboration de ces modèles qui sont répartis entre les joueurs :

  Quand le propriétaire du Salon ouvre le salon, son EJB Statefull MODELE est créé. Il initialise le modèle par exemple le jeu de carte et le mélange (et d'autres informations). Il met à jour la BD pour désigner que le Salon est ouvert ce qui permet pour un joueur qui se connecte de voir quels sont les salons ouverts et d'en choisir un.

  Quand un joueur rejoint le salon, son EJB Statefull MODELE est créé . Le MODELE notifie son existence. L'EJB du propriétaire est prévenu ainsi que tous les autres joueurs.

  Tous les autres joueurs rejoignent de la même façon le salon.

  Quand la page du navigateur de la Table de POKER est chargée, la VUE est créée. Elle s'initialise en fonction de l'état du MODELE.

  Quand le propriétaire du salon fait l'action de démarrer la partie, son modèle calcule le joueur qui doit commencer à jouer. L'EJB du propriétaire change son état du modèle qui est notifié à tous les autres modèles.

  Quand ce joueur réalise l'action, son EJB reçoit son action, fait le traitement correspondant qui met à jour le joueur courant dans son modèle et notifie le nouvel état du modèle à tous les autres joueurs. Le joueur suivant peut ainsi jouer. Et ainsi de suite.

En conclusion :

On peut voir ici la simplicité (en terme de composant) de cette architecture en utilisant une communication à base de message. Ici, inutile de gérer des EJB Singleton (qui est toujours compliqué à gérer). Bien adapté pour des SI de dimensionnement moyen.

Elle est très évolutive.

En résumé, au lieu de centraliser l'information dans un composant, elle est répartie dans les composants de chaque utilisateur et la cohérence de cette information est réalisée par l'envoi de message.