

Thomas Buisson
Département Informatique
INSA de Lyon

Porphyry 2000

Module Client Serveur

Sommaire

1. INTRODUCTION	2
2. ARCHITECTURE	3
2.1. Contraintes d'implémentation.....	3
2.2. Architecture technique	3
3. CONCEPTION GENERALE.....	6
3.1. Description	6
3.2. Cas d'utilisation	6
4. AMELIORATIONS ET EVOLUTIONS ENVISAGEABLES :.....	7
4.1. Vitesse de traitement des requêtes :	7
4.2. Sécurisation des connexions.....	7
4.3. Répartitions en stages – PFE.....	8

1. Introduction

Ce dossier concerne le développement du second prototype porphyry.

Le prototype réalisé est fonctionnel, mais quelques limites sont apparues lors de la dernière semaine de développement, notamment dans certains cas particuliers.

Ce document est un résumé des quelques choix de conceptions retenus.

Il présente néanmoins dans une première partie la conception et de la réalisation du prototype et dans une seconde partie les évolutions possibles, les améliorations que l'on peut y apporter.

La troisième partie est une proposition de répartition du travail nécessaire pour obtenir une version "finale" du prototype, du moins au niveau des serveurs, en un PFE, ainsi qu'un stage (3 ou 4 IF).

2. Architecture

2.1. Contraintes d'implémentation

Les contraintes d'implémentations sont liées a la sécurité des données et à la vitesse de traitement des requêtes des utilisateurs.

- Sécurité des données :

l'application qui tourne sur le serveur doit veiller a la cohérence entre les données, notamment entre les données privées modifiables, les données privées non modifiables, et les données publiques. Le fait que plusieurs utilisateurs travaillent sur les mêmes données augmente d'autant plus les risques d'incohérences.

- Vitesse d'exécution :

Le critère principale d'appréciation du prototype par les utilisateurs, après l'ergonomie, est comme pour tous les moteurs de recherche, la vitesse de traitement des requêtes. La structure retenue sera donc optimisée pour accélérer les traitements, l'occupation mémoire n'étant pas du tout considérée comme une contrainte.

2.2. Architecture technique

L'architecture retenue est de type trois tiers :

- Le client :

Sa conception est traitée en détail dans le document de conception qui le concerne (Cf Aurelien Benel et Mehdi Lababidii)

- Le serveur :

La conception du serveur vise principalement a répondre aux deux contraintes de vitesses et de cohérence.

Le moteur du serveur est le package model, développé pour le premier prototype porphyry. (Cf Rapport sur le 1er prototype)

L'interface avec le client est réalisée via RMI. Un manager implémente l'interface publique, afin de gérer les identification des utilisateurs (login) mais aussi le chargement en mémoire des données propres à chaque client lors de la réception d'une requête. (ou si la mémoire est suffisante et les données déjà présentes, le passage au model de l'adresse de ces données).

La structure retenue pour la gestion des données est de type mixte mémoire / base de données. En effet une partie des données, publiques ou privées, peut être considérée comme publiée, ou définitive. Apres avoir calculé la fermeture transitive de ces données (opération très lente), on peut stocker ces dernières directement sur la bd, les traitements correspondant étant alors des requêtes sans récursivité.

Pour la partie non définitive des données, la contrainte de cohérence nous interdit de les traiter de la même façon, car il faudrait recalculer la fermeture transitive après chaque modification, ce qui n'est pas envisageable. le traitement effectué est donc un parcours récursif de l'arbre, stocké en mémoire dans un partial order (Basé sur des hashmaps)

On implémente ainsi un broker permettant de répartir pour chaque itération la requête sur tous les partials orders concerné (privé client, groupe client, et public).

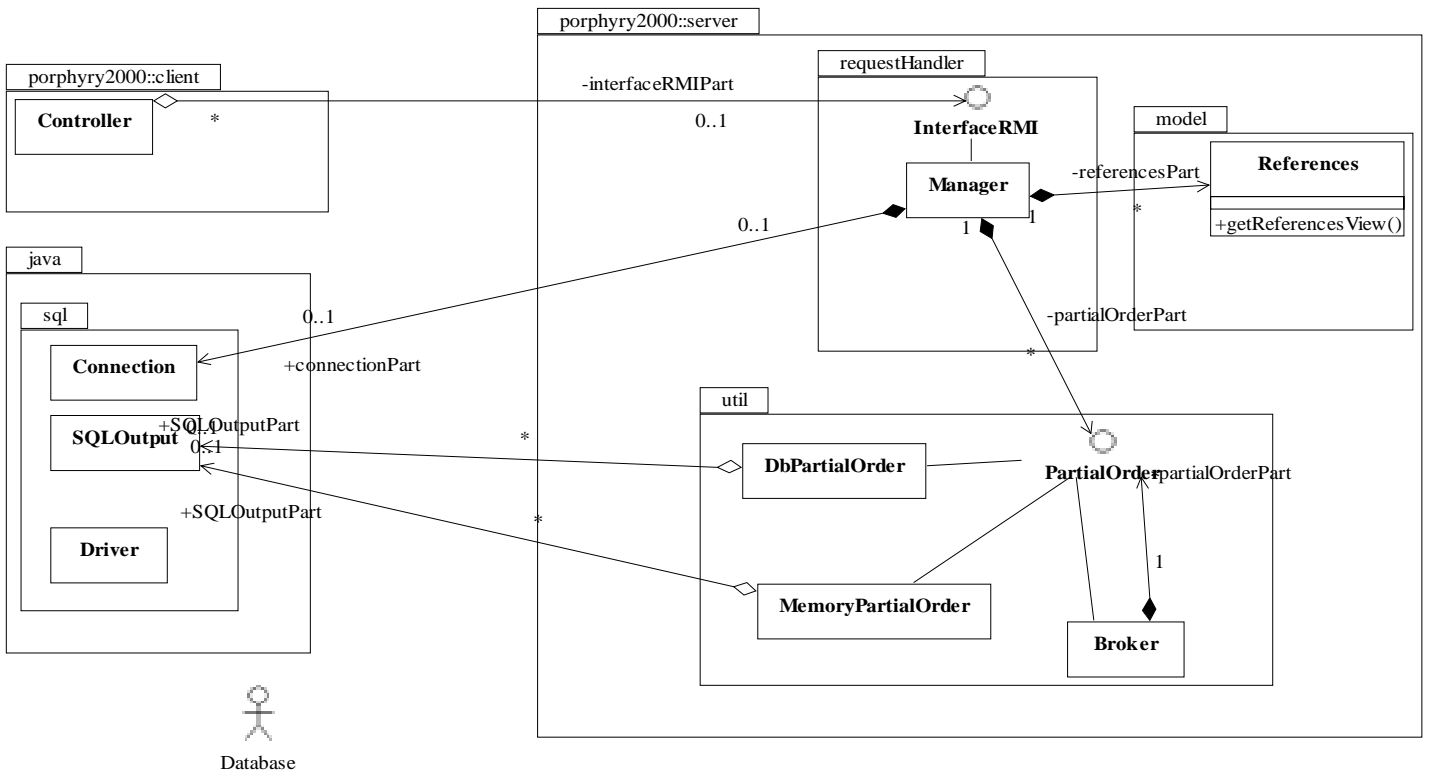


Diagramme de classe

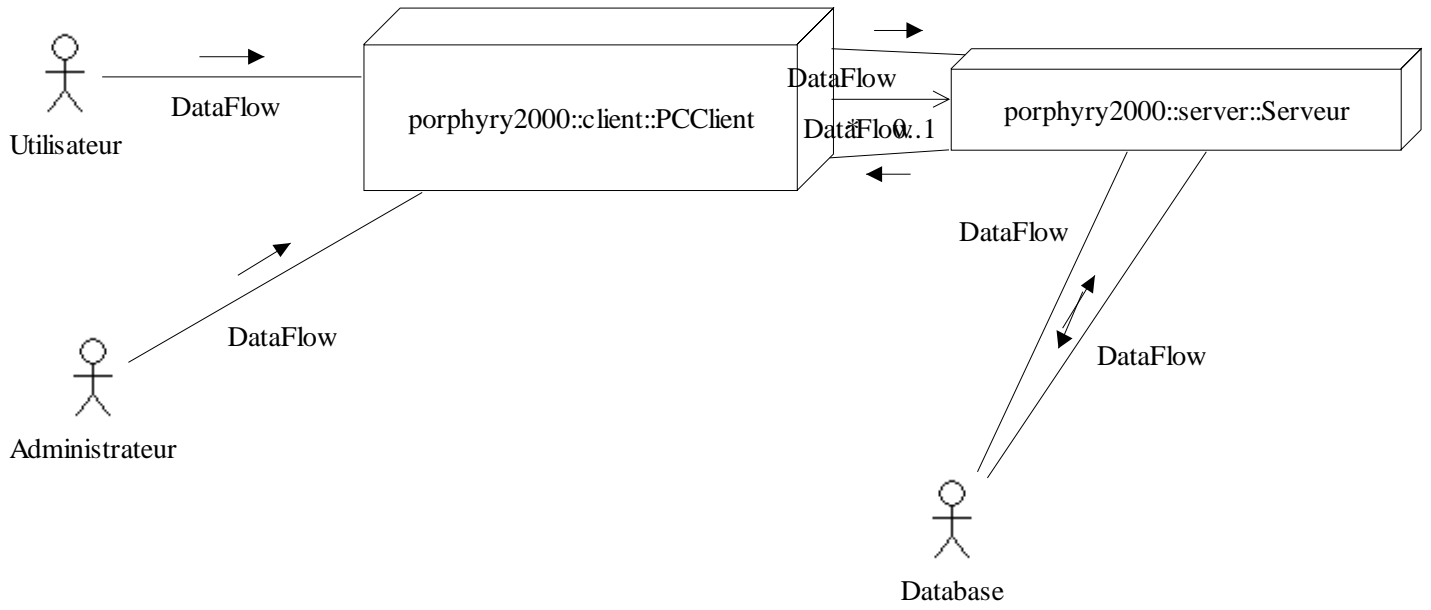


Diagramme de déploiement

3. Conception générale

3.1. Description

La conception générale vise principalement à répondre aux deux contraintes principales, à savoir vitesse d'exécution et la cohérence des données.

Ces deux contraintes ont un impact particulier sur l'architecture du système, notamment au niveau de l'implantation en mémoire des données, qui se répercute sur tout le reste de l'application.

Cette conception est réalisée dans un objectif de développement en java, pour réaliser une application client serveur d'architecture 3 tiers. on utilise les couches RMI et JDBC pour assurer les communication entre les différents tiers.

Le client n'est pas traité dans ce document de conception.

Le serveur peut être découpé en trois couches distinctes. La première concerne l'interface fournie au client, le traitement de toutes les requêtes reçues via la couche RMI. La seconde peut être assimilée au moteur de l'application, et gère toute la partie calculatoire. La troisième concerne la communication avec la ou les base de donnée, ainsi que le chargement de ces dernières en mémoire.

3.2. Cas d'utilisation

On peut distinguer principalement 4 cas d'utilisation du prototype :

- la consultation anonyme : Le client consulte la partie publique du graphe et ne dispose pas de droit d'écriture (ajouts de descripteurs et de spécialisations)
- la consultation en lecture / écriture : le client dispose d'un graphe personnel accessible en écriture.
- la publication (administration de groupe) : le client dispose des droits de créations de graphe, de fusions entre les graphes, et de publication d'une portion de graphe.
- la maintenance (administration générale) : le client dispose des droits de création de groupe.

4. Améliorations et évolutions envisageables :

Après le développement du 2nd prototype, plusieurs évolutions et améliorations apparaissent comme envisageables, voire même comme nécessaires.

Ces évolutions concernent plusieurs aspect du prototype, notamment la vitesse de traitement des requêtes, la sécurité des connexions, la répartition du serveur sur plusieurs machines.

4.1. Vitesse de traitement des requêtes :

Le module GraphHandler (partie moteur du serveur) est très gourmand en CPU/mémoire. Afin d'améliorer ses performances, plusieurs solutions complémentaires peuvent être mises en place.

La première est la répartition du serveur sur plusieurs machines. (grappe de PC). Un serveur de tête répartissant les requêtes a plusieurs autres serveurs, en fonction de l'utilisateur qui effectue cette requête, du type de requête, ou de divers autres paramètres tels que la charge des serveurs.

La seconde optimisation, complémentaire à la première et pouvant être traitée par la même personne est la création d'un cache sur les requêtes effectuées. En effet, pour un partialorder donné, certaines requêtes reviennent relativement souvent, surtout lors de la consultation de la "tête" du graphe. Une étude peut être envisagée pour définir le niveau du prototype auquel devront être implantés ce ou ces caches de requêtes.

Une troisième amélioration du temps de calcul pourra être obtenue en optimisant directement les requêtes effectuées sur la base de donnée, en fonction de la base. En effet, la version actuelle du prototype se veut indépendante du SGBD utilisé. Les requêtes sont donc en SQL standard, alors qu'il existe de nombreuses optimisations sur les différents SGBD (par exemple, l'utilisation de requêtes précompilées avec paramètres). On peut donc envisager un système équivalent à la "localization factory" mise en place pour traduire les messages d'erreurs dans une langue voulue. Les accès aux bases de donnée se feraient alors en passant par une couche optimisée en fonction de la base utilisée.

Toujours pour améliorer les performances du serveur, nous avons pu noter que la structure retenue s'est montrée quelque peu limitée par l'un des calcul (récuratif) qui n'a pas été optimisé dans la base de donnée. (c'est le parcours du graphe, étage par étage, qui détermine l'état des descripteurs). Il existe plusieurs solutions pour résoudre ce problème.

La première est une modification de l'algorithme, afin de récupérer un étage en une seule requête (chercher les fils des descripteurs avec un temps d'avance), la seconde est d'utiliser pour le DBPartialOrder la même structure en mémoire que pour le MemoryPartialOrder. Cette solution augmenterait l'espace mémoire occupé, mais serait la plus optimisée.

4.2. Sécurisation des connexions

A un niveau plus éloigné du noyau de l'application, il reste à développer un utilitaire de connexion sécurisé, avec cryptage des passwords, en s'appuyant sur les résultats du stage de Vanessa Auzanne par exemple.

En effet, la version actuelle du prototype utilise un login et un password pour connecter les utilisateurs, puis un numéro de session pour reconnaître le propriétaire d'une requête. Cette solution est acceptable dans le cadre d'un prototype de démonstration, mais n'est en rien sécurisée. Il suffit en effet de se connecter avec un nom d'utilisateur correct, puis de donner

un numéro de session correspondant à un autre utilisateur (qui doit cependant être logué à cet instant) pour avoir accès à toutes ses données (en lecture et écriture).

4.3. Répartitions en stages – PFE

A mon avis, la partie “ parallélisation ” du serveur sur plusieurs machines, avec gestion d’un ou plusieurs caches, au niveau des requêtes envoyées par l’utilisateur, peut tout a fait convenir pour un sujet de PFE.

La création d’une couche de connexion sécurisée et son implémentation entre le client et les différents serveurs (graphe, document) peut être traité par un stagiaire 3 ou 4IF. Les 4 ifs ont cependant de meilleurs connaissances concernant la sécurité.

En fonction de la durée de son stage, le stagiaire pourra aussi s’occuper de l’optimisation poussée des accès a la base de donnée.

Les sujets pourraient donc être les suivant :

STAGE : conception et mise en place d'un système de gestion sécurisé de connexions dans le cadre du développement d'un outil de consultation de documents.

PFE : conception et développement de la version distribuée (multi-serveurs), implémentant une gestion de caches de requêtes, de l'outils de consultation de documents pour experts appliqué à la chronique des fouilles, à partir d'un prototype conçu selon une architecture 3 tiers.