Le génie pour l'industrie

Département de génie logiciel et des TI

Rapport de laboratoire

Cours	LOG735
Session	Eté 2015
Groupe	01
Laboratoire	LABORATOIRE 2 – Transmission synchronisée d'événements entre applications
Chargé de laboratoire	Thierry Blais
Professeur	Lévis Thériault
Étudiant(e)(s)	Moreau Max Charly Simon
Adresse(s) de courriel	max.moreau.1@ens.etsmtl.ca charly.simon.1@etsmtl.net
Code(s) permanent(s)	MORM30038905 SIMC28069108
Date de remise	3 juin 2015

1 Tables des Matières

1 Tables des Matières
2 Introduction
3 Réponses aux questions
    3.1 Question 1)
        Transparence :
        Performance :
    3.2 Question 2)
    3.3 Question 3)
    3.4 Question 4)
4 Diagrammes
    4.1 Diagramme Synchronisation :
5 Conclusion
6 Annexes

2 Introduction

Ce laboratoire sert a nous familiarisé aux différents soucis inhérents a la synchronisation de système distribue. Nous expérimentons ceci par une approche pratique du problème avec un cas concret a résoudre.
Nous discuterons des approche alternatives de notre solution retenu par une analyse sur les différents attribues de qualité logicielle.

3 Réponses aux questions

3.1 Question 1)

De façon générale, expliquez les différents défis à relever par rapport à la synchronisation (de messages et/ou d'applications) et parlez brièvement des solutions possibles pour chacun des défis mentionnés.

Nos messages sont dans le cadre d'un système distribue de fait il doivent répondre au même critères / défis des systeme distribue a savoir : Hétérogénéité, Ouverture, Sécurité, Extensibilité (scalability), Gestion des défaillances, Concurrence , Transparence.
Nous aborderons les 4 points que nous avons juger le plus important pour cete réalisation a savoir : Transparance, Extensibilité, Fiabilite (defaillance) et Securite.

Transparence :

Un des défis majeurs a relever dans un système distribue est relie a un soucis de transparence; l'utilisateur ne doit pas avoir se soucier de la synchronisation des clients et des serveurs. Les utilisateurs doivent percevoir un système unique, peu importe les messages communiqués ou le nombre d'applications connectées.

Ces problèmes pouvant nuire a la transparence des services peuvent être résolues par l'intermédiaire d'interfaces de communications communes dont l'implementation est adaptée a la machine sur laquelle la ressource sera récupère ou exécuté. L'architecture logicielle *Corba* en est un exemple, car elle permet une abstraction de la communication entre les différentes machines du système en fournissant des services communs par l'intermédiaire d'interfaces de communication.

Il est également possible de définir son propre protocole de communication. Cette seconde s'avère être une solution permettant une définition optimisée et complète des messages nécessaire au bon fonctionnement du système (dans note contexte). On réduit ainsi la complexité inhérente a *Corba* tous en gardant l'abstraction faite au niveau des différents noeuds.

Performance :

Un autre défis important est celui de la performance du système. La vitesse de traitement ne dois pas diminuer en fonction du nombre d'application ou du moins dans des seuils tolérable (10%) autrement notre application n'est pas extensible et l'on perd de l'intérêt a l'utilise. L'extensibilité du système permet d'accroître la capacité de traitement et donc d'être plus performant pour le traitement d'un grand volume par rapport a une solution unique. Néanmoins, si les messages de synchronisation prennent trop de temps a être transfère, on réduit notre temps réel de traitement et perd en efficacité.

Une solution envisageable serait de ne synchroniser que les messages critiques réduits a leur minimum. Ceci permettrait de minimiser la taille des données a transférer et a traiter, permettant par la même occasion d'obtenir une plus grande disponibilité du mode de communication et une plus grande rapidité du traitement des encapsulations et desencapsulation des messages.

Fiabilité :

Un attribut de qualité supplémentaire qui ne doit pas être négligé dans la réalisation d'un système distribue est la fiabilité des noeuds. Si un de celui-ci entre dans un état non-disponible ou isolé, le bon fonctionnement du système peut en être altéré. Ceci pourrait engendre une perte de donnée critique au système. De plus la synchronisation avec le reste du système doit être rétabli lorsque le noeud se réintègre au réseau. De nombreux message de synchronisation peuvent alors être générer afin de rétablir l'état du système.

Afin de parer a ce problème, il peut être envisageable de dupliquer l'information ou le traitement sur plusieurs noeuds. Néanmoins, cette solution a pour conséquence de diminuer la disponibilités des noeuds. Une autre alternative serait d'effectuer des petits traitements rapides afin de diminuer la probabilité qu'un noeud ne puisse pas terminer sa tache en cour ainsi que de minimiser la taille de données a synchroniser.

Securite

Les messages étant transmis sur un bus toute les applications connecte y ont accès. De même si une application émet du mauvais contenue toute les applications y seront sujet et pourrais toute crasher.

Afin de parer a ce problème nous pourrions authentifier les applications se connectant au bus avec une clé. Remettant ainsi la responsabilité du bon fonctionnement de l'ensemble au sérieux de la gestion des clé. Nous pourrions aussi encrypter nos différents messages.

Bien que nous avons identifier ce volet comme important pour les messages nous ne n'y attarderons pas dans sa réalisation car nous contrôlons l'ensemble de l'environnement pour ce laboratoire.

3.2 Question 2)

Expliquez l'approche que vous avez prise pour permettre l'envoi synchronisé des messages. Expliquez ensuite les modifications apportées à chaque classe (création de classes, modification des attributs et des méthodes.

Referez vous a l'annexe pour la liste exhaustive des modifications.

Nous avons modifier l'interpreter de chaque application pour définir des règles de comportement quand ils reçoivent un message de synchronisation.
En résume nous avons définit un protocole pour les message que les applications doivent suivre. Ceci permet d'avoir une solution indépendante du nombre d'application a l'écoute et robuste en ayant plus de transparence, d'extensibilité et de et de fiabilité.
Ce protocole est diviser en deux parties : le message de début de transaction et le message de ack.
Celui ci est conçu comme suis :

Message de début de transaction

SeqNumber	TransactionId	totalAck	Message

- Le SeqNumber permet didentifier et d'ordonner les message recus
- Le transactionId permet d'identifier les groupe de transaction
- Le TotalAck est le nombre de ack que le client doit attendre avant d'afficher son message

Message de Ack

SeqNumber	TransactionId	ACK_Type

- Le SeqNumber permet didentifier et d'ordonner les message recus
- Le transactionId permet d'identifier les groupe de transaction
- Le ACK_Type permet de differentier un ack standard d'un ack de fin de transaction

A noter que les messages que chaque application devait afficher, était définit en dur lors de leur initialisation. Nous nous somme permis de modifier ce message pour simuler notre processus car nous estimons que réellement le message synchroniser devrait être transmit par un des noeuds et non initialiser de façon statique.
L'initialisation actuelle cassant directement tous intérêt du distribué car le message étant dépendant du nombre d'application.

3.3 Question 3)

Discutez des améliorations possibles à effectuer à votre implémentation en fonction des notions d'extensibilité et de performance : que devriez-vous corriger pour que l'envoi de messages synchronisés fonctionne avec un nombre variable d'applications?

Une amélioration possible dans notre implementation qui permettrait une extensibilité complète, c'est a dire permettre au système de fonctionner si un noeud n'est plus présent , serait de transmettre le message au complet a toutes les applications du système a la place du nombre total de ack. La fiabilité du système serait également améliorer en ajoutant une temporisation permettant aux applications de prendre la relevé si une d'entre-elles n'est plus intégrée au système pour quelques raisons que ce soit.

Néanmoins, une perte de performance sera a considérer. Plus la chaîne de caractères a afficher sera grande, plus le temps de temps de transmission initial sera élevé.

3.4 Question 4)

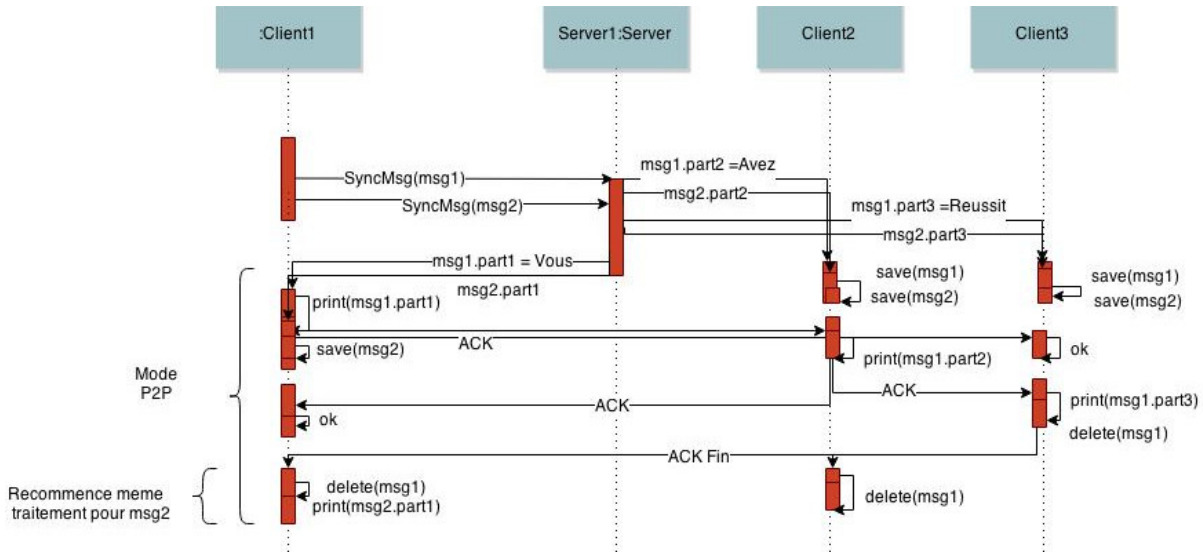
Discutez d'au moins une solution alternative qui aurait pu répondre à la tâche demandée ; pensez à comment les responsabilités auraient pu être différemment assignées entre les applications et le bus d'événements

Notre solution retenu est robuste et non dépendante du nombre de client mais est dépendante de l'implementation du client. Si celui-ci ne respecte pas le protocole alors l'ensemble de la chaîne de traitement peut être brise.
Déplacer la logique sur le serveur évite ce cas et nous assure une certaine sécurité au niveau du traitement mais nous rend dépendant de l'exécution du serveur. Si celui-ci crash la séquence est perdu. On peut bien sur prévoir des backup mais c'est une solution assez centralise, car c'est cette centralisation qui nous assure le sequencement des opérations.
C'est pourquoi nous avions choisit la solution plus décentralisé, nous parlons d'une architecture P2P vs Client-Serveur.

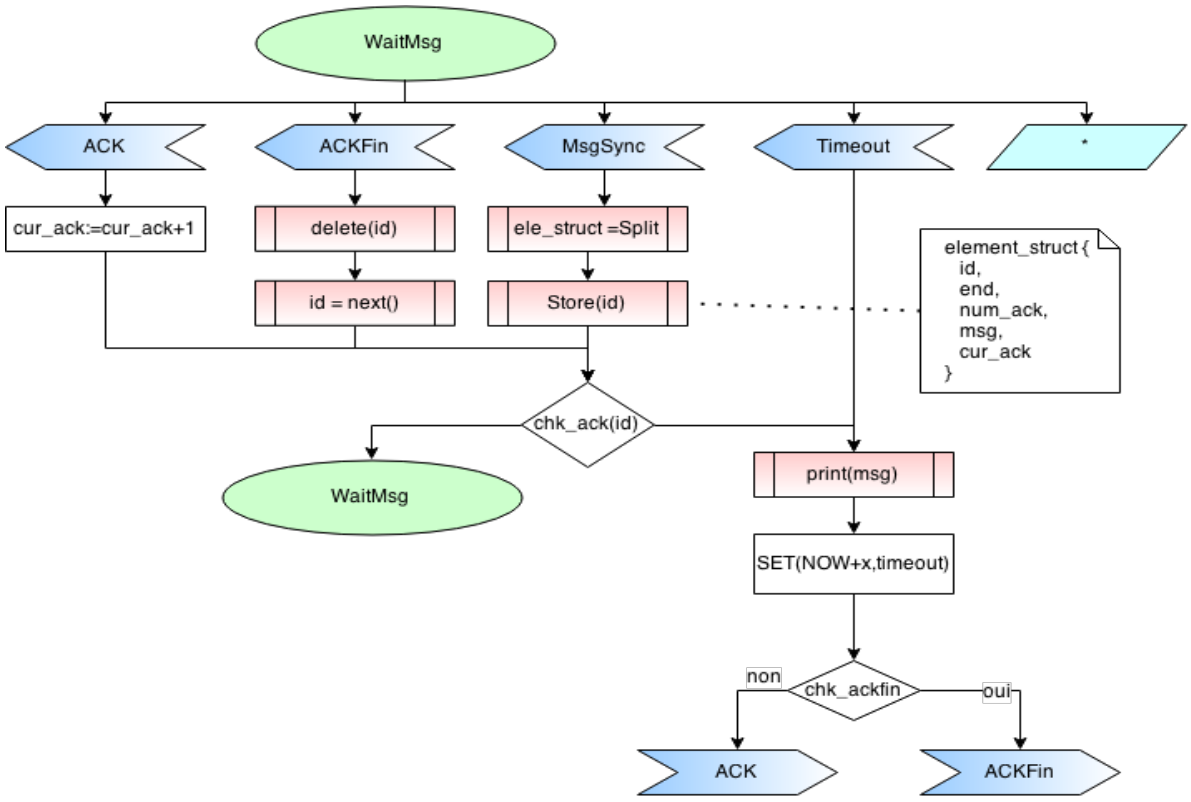
4 Diagrammes

4.1 Diagramme Synchronisation :

Mode P2P



4.2 SDL Protocole :



5 Conclusion

Ce laboratoire nous a permis de créer une petite architecture distribue sous la forme d'un serveur-serveur, avec plusieurs instances devant se synchroniser entre eux afin de fournir un service adéquat. Ces différents serveur peuvent être vus comme différents services sur une architecture distribue qui pour fournir un cas d'utilisation correcte devrait se synchroniser entre eux.
Nous aurions également pu choisir une architecture plus classique de client-serveur mais nous l'avons estimer moins intéressante.
Le traitement en question des différents service est comme pour le laboratoire 1, dérisoire par rapport aux temps de synchronisations. (Il s'agit d'un simple affichage de message). On simule toutefois un temps de traitement différents par client ce qui permet de se rapprocher d'un cas d'utilisation réel. Nous déplorons toutefois que les délais de transmission ne soit pas simuler également.

6 Annexes

Cette annexe présente les changements entres les différentes questions dans le code de façon normalisé.
Nous utilisons la commande linux : ‘git diff $INIT_SRC HEAD -- src ’ pour ceci.
Avec $INIT_SRC la valeur du hash correspondant a l’initialisation de nos src dans notre dépôt git.
Ceci nous produit l’ensemble des changements dans le dossier src depuis l’initialisation a HEAD (maintenant).
Ces changements explicite sont disponible dans le dossier ‘doc’ dans le fichier ‘fullsrc.diff’