**Master Recherche**
**Informatique de Lyon**

# MRI - RTS6
# ~
# Architectures et Middlewares pour les Collecticiels

René CHALON, Ecole Centrale de Lyon

Plan:



**Version du 20/12/2006**

1

## 1 - Introduction

- Un collecticiel est plus complexe à mettre en œuvre qu'une application mono-utilisateur :
  - Nouvelles problématiques liées au trèfle fonctionnel :
    - Communication: comment upporter la communication entre les utilisateurs
    - Coordination: comment arbitrer les conflits entre les utilisateurs
    - Production: comment gérer une production commune
  - Assurer un système cohérent face aux actions concurrentes des utilisateurs répartis dans l'espace et dans le temps
  - Prendre en compte l'infrastructure de communication: ses caractéristiques (débit, délai) et des défaillances
- Solutions:
  - Modèles et architectures
  - Outils logiciels: bibliothèques, *middleware*, *framework* …

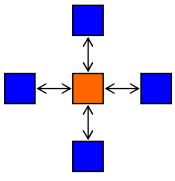
## Modèle de processus de Groupe

- 3 types de processus de groupe (Cf. Rapaport 1991):
  - Modèle de processus centralisés
    - L'information est stockée et gérée dans un système centralisé
    - Tous les utilisateurs partagent les mêmes informations
  - Modèle de processus distribués non-répliqués
    - Les processus et les données sont répartis sur les différentes machines du réseau mais n'existent qu'en un seul exemplaire
  - Modèle de processus distribués et répliqués
    - Les processus et les données sont répartis et répliqués sur les différentes machines du réseau
    - Plus résistant aux erreurs (pannes de machines, pannes réseau)
    - Temps de réponse plus court

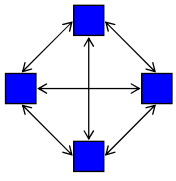


## Modèles de Communication

- Type de distribution de l'information:
  - Un à un
  - Un à plusieurs
  - Plusieurs à un
  - Plusieurs à plusieurs
- Mécanisme de transport:



  - Communication via une unité de contrôle centrale:
    - N liens pour N nœuds
- Sens de transmission:
  - Unidirectionnel
  - Bidirectionnel



  - Communication directe:
    - N(N-1)/2 liens pour N nœuds

## Contrôle de la concurrence (1/2)

- Aspects de la concurrence:
  - Temps de réponse: dans les collecticiels synchrones le temps de réponse doit être court ➔ Pb des WAN
  - Interface utilisateur : les interfaces ne doivent pas montrer des états « trop vieux » : WYSIWIS (strict ou relâché)
  - Réplication de l'information : permet de réduire les temps de réponse mais nécessité de garder la consistance
    - Alternatives:
      - Réplication de l'interface utilisateur
      - Propagation des opérations qui sont exécutées de la même manière sur toutes les machines
  - Robustesse: reprise après une panne machine ou une erreur réseau.
  - Notifications: alerter les utilisateurs en cas de changements conflictuels



## Contrôle de la concurrence (2/2)

- But : Conserver la consistance de l'information :
  - Consistance stricte:
    - tous les états intermédiaires doivent identiques pour tous les utilisateurs
  - Consistance lâche:
    - seul un état final identique pour tous les utilisateurs est suffisant, les états intermédiaires pouvant être différents.
- Moyens pour conserver la consistance :
  - Détection des conflits de concurrence d'accès à des données
  - Résolution des conflits:
    - Verrouillage a priori des données (peu efficace, souvent bloquant)
    - Opérations en conflits sont annulées (nécessite la réversibilité des opérations)
    - Fusion des opérations (difficile car dépend trop de la sémantique de l'application)

## Rôle des membres du groupe

- Rôles:
  - Les rôles correspondent à des fonctions sociales
    - → Basés sur des savoir-faire, compétences et connaissances
  - Un rôle définit des droits et des devoirs au sein du groupe
    - → Droits d'accès sur les données (aucun, lecture, écriture, lect/écriture)
    - → Activités à accomplir pouvant être formalisées par un workflow
  - Rôles définis de manière formelle ou informelle
- Un utilisateur a un ou plusieurs rôles
  - Exemple: un chef de projet est aussi un programmeur d'une équipe
  - Problème éventuel de contradiction entre les rôles



2

## 2 - Architectures des collecticiels

- Architecture centralisée:
  - L'application s'exécute sur une seule machine centrale, seul les interfaces utilisateur sont distribuées:
    - → Partage d'écran/de fenêtre; avantage permet de rendre « collaborative » toute application mono-utilisateur
    - → Interfaces dédiées à chaque utilisateur communicant avec un serveur central (exemple des système de messagerie instantané ou de *chat*)
    - → Partage d'information: pour système asynchrones comme des forums ou des wiki
- Architecture répliquée:
  - Les informations et les applications sont répliquées sur chaque machine des utilisateurs:
    - → Les actions des utilisateurs doivent être transmises à tous les réplicas où elles doivent être sérialisés et traitées par l'instance locale de l'application

## Autres architectures

- Classification de Philips:
  - Centralisé: application sur un serveur
  - Répliqué: l'application est entièrement répliquée sur toutes les machines
  - Coordination centralisée: architecture répliquée mais avec mécanisme centralisée de maintien de la cohérence
  - Semi-répliqué: certaines parties sont répliquées, d'autres centralisées
  - Systèmes flexibles: on peut spécifier l'architecture de réplication indépendamment du comportement du système
  - Systèmes dynamiques: systèmes flexibles dont l'ajustement de l'architecture de réplication peut se faire pendant le fonctionnement du système

R. CHALON MRI-RTS6 9

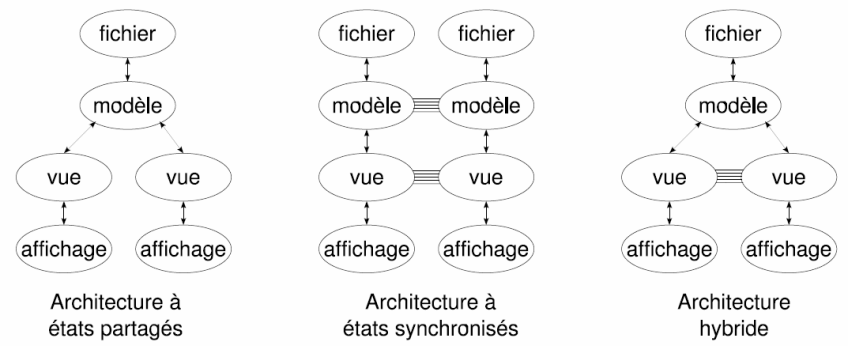
## Architectures répliquées

- Caractéristiques des architectures répliquées:
  - Les états initiaux des réplicas peuvent être différents
  - Comportement déterministe: si les états initiaux sont identiques et que la séquence des entrées est identique les réplicas doivent générer les mêmes sorties et terminer dans le même état final.
  - État identique: toutes les applications ne sont pas strictement dans le même état selon les préférences des utilisateurs
  - Ordre des événements d'entrée: la consistance impose (en général) que les événements d'entrée soient dans le même ordre pour tous les réplicas. Cela peut être assuré par la numérotation des actions et un mécanisme de « passage de main »
  - Appartenance à une session: des participants peuvent partir ou joindre une session en cours. Dans le dernier cas, il faut soit rejouer la séquence des actions soit dupliquer l'état d'un autre réplica pour que le nouveau membre ait une application à jour.

R. CHALON MRI-RTS6 10

## Taxonomie de Patterson

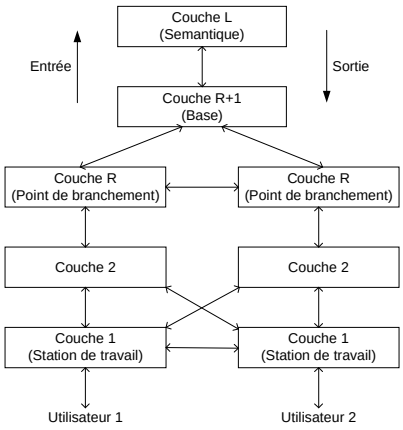
- 3 exemples d'architecture :





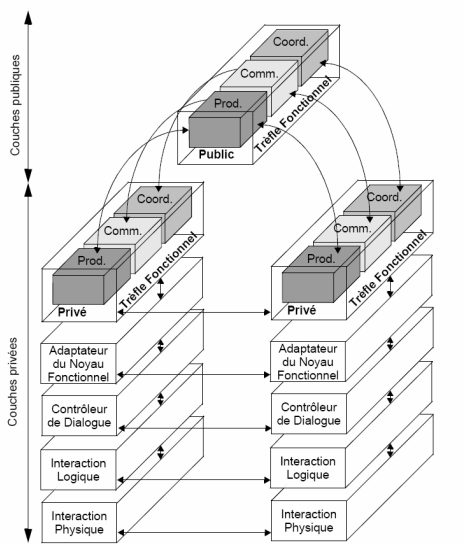

## Modèle de Dewan

- Modèle d'architecture générique de Dewan
  - Nb arbitraire de couches:
    - → R couches répliquées
    - → L-R couches centralisées
  - Communication entre couches successives
  - Communication entre éléments répartis dans la couche



## Modèle de Laurillau

- Modèle Clover:
  - Synthèse entre:
    - Modèle Arch
    - Modèle de Dewan
    - Trèfle fonctionnel du collecticiel






3

## 3 - Systèmes distribués

- La plupart des principes des systèmes distribués jouent un rôle important pour les collecticiels, surtout dans les architectures répliquées

- Rappel de ces différents principes et concepts :
  - Transparence
  - Mécanismes de communications :
  - Client-serveur
  - RPC
  - Systèmes distribués orientés objet
  - Applications distribuées

## Transparence

- Transparence vis-à-vis de l'utilisateur du système,
  - le système distribué apparaît comme un système centralisé
- Différents types de transparence:
  - Transparence de localisation: une ressource réseau est accessible qu'elle soit sa localisation physique (exemple impression réseau)
  - Transparence de réplication:
    - Accès à n'importe quelle réplica d'une ressource répliquée
  - Transparence à la concurrence:
    - Le contrôle de la synchronisation est incluse dans le système distribué
  - Transparence à la défaillance:
    - par la réplication des ressources
  - Transparence à la migration:
    - Une ressource est déplacée d'une machine à une autre
    - Une machine est déplacée d'un réseau à un autre (mobilité):
      - Migration off-line: ordinateur portable connecté dans différents réseau
      - Migration on-line: téléphone mobile changeant de cellule (*roaming*)

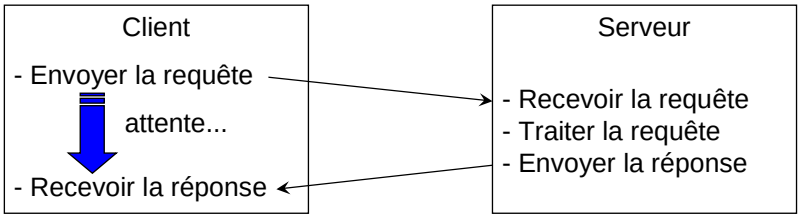


## Mécanismes de communication

- Partage d'information:
  - Communication entre composants assurés par un système de fichiers partagés ou une base de donnée
- Échange de messages:
  - Similaire aux IPC (*inter-process communication*)
    - Message = en-tête (Id émetteur, Id récepteur, type message) + corps
  - Le message peut être:
    - Asynchrone: l'émetteur envoie et continue
    - Synchrone: l'émetteur se suspend jusqu'à ce que le récepteur retourne un accusé de réception
- Producteur-consommateur:
  - Similaire aux *pipes* Unix
  - Le producteur produit des messages qui sont consommés de manière asynchrone par le consommateur

## Modèle client-serveur

- La communication est synchrone:
  - le client est bloqué tant qu'il n'a pas reçu la réponse du serveur




- Dans ce modèle, le serveur traite une requête à la fois:
  - pour éviter la perte des requêtes venant d'autres clients:
    - → utilisation d'une file d'attente des messages côté serveur
  - pour optimiser le temps de réponse:
    - → un processus attend les requêtes ; quand une requête arrive un nouveau processus (ou *thread*) est lancé et traite la requête

R. CHALON MRI-RTS6 17

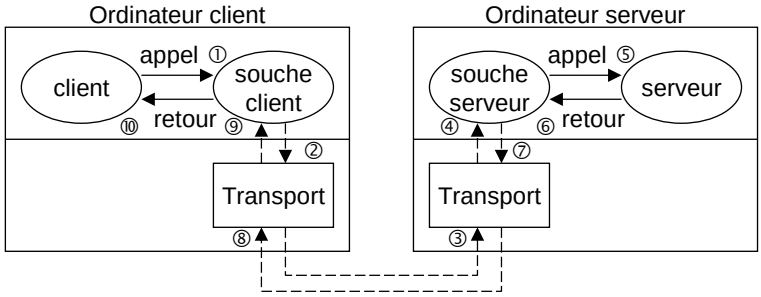
## RPC : Appel de procédure à distance

- RPC [Remote Procedure Call]:
  - appel de procédure à distance
  - extension du concept d'appel de procédure locale :
    - → invoquer une procédure avec des paramètres en entrées
    - → la procédure s'exécute sur une autre machine
    - → les résultats sont disponibles par des paramètres de sorties
- Intérêts:
  - c'est une réalisation du mode client-serveur
  - plus simple à utiliser que le "mode message" fourni par les Sockets
- Inconvénients:
  - pas de possibilité de messages initiés par le serveur (sauf mécanisme de call-back)

R. CHALON MRI-RTS6 18

## Principe de transparence des RPC

- Transparence vis-à-vis des applications:
  - assurée par des procédures spéciales appelées "souches" (*stub*) du côté client et du côté serveur (parfois appelées "talons")
  - les souches fonctionnent comme des représentantes locales des procédures
  - seules les souches savent que l'appel est destiné à une autre machine




R. CHALON MRI-RTS6 19

## Systèmes distribués orienté objet

- Objets distribués:
  - Les objets sont répartis sur différentes machines
  - Les méthodes sont appelées à travers le réseau de manière transparente : RMI=Remote Method Invocation (Cf. RPC)
- Migration des objets:
  - Possibilité pour les objets de passer d'une machine à une autre (transparence de migration)
  - Localisation des objets sur les machines : notion de Broker
- CORBA:
  - Common Object Request Broker Architecture
  - Architecture standardisé par l'OMG

R. CHALON MRI-RTS6 20

## Applications distribuées

- Application distribuée:
  - Découpée en un ensemble d'unité fonctionnelles qui coopèrent et interagissent. Chaque unité fonctionnelle possède un état interne et des opérations pour modifier cet état
  - Chaque unité fonctionnelle peut être assignée à une machine différente
  - Les unités fonctionnelles communiquent entre elles par échange de messages, RPC, ou des structures de données partagées



4

## 4 - Contrôle de la concurrence

- Comme dans les systèmes distribués mais:
  - Granularité plus grande des données
  - Données structurées
  - Gestion de haut niveau :
    - → l'utilisateur peut être directement impliqué dans la gestion de la concurrence ou la résolution des conflits et des inconsistances
- Classification:
  - Contrôle optimiste : pas de garantie que les données partagées soient consistantes à tout instant
  - Contrôle pessimiste : nécessaire lorsqu'il faut garantir la consistance des données :
    - → Contrôle centralisé
    - → Contrôle décentralisé
      - Avec vote
      - Sans vote

## Contrôle optimiste

- Pas de contrainte de l'utilisateur:
  - À tout instant, tout utilisateur peut modifier des données
- Au moment d'enregistrer les modifications, on vérifie que d'autres utilisateurs n'ont pas fait des modifications de manière concurrente :
  - Si pas d'autres modifications :
    - → on enregistre les modifications de l'utilisateur
  - Si il y a eu des modifications concurrentes :
    - → on prévient l'utilisateur du conflit
    - → on enregistre ses modifications dans une copie.
    - → L'utilisateur peut ensuite fusionner manuellement ses modifications avec le dernier état des données.

R. CHALON MRI-RTS6 23

## Contrôle centralisé

- Par contrôleur :
  - Même si les données sont répliquées, un seul site est le responsable des données en écriture
  - Avantage : simple à mettre en œuvre
  - Inconvénients :
    - → Goulot d'étranglement si de nombreux participants
    - → Si le contrôleur tombe en panne, il faut élire un nouveau contrôleur, ce qui est particulièrement délicat
- Par jeton (*token*) :
  - Un jeton circule sur un anneau virtuel des sites;
    - → seul le site qui détient le jeton à l'accès en écriture aux données.
    - → Il passe le jeton au suivant au bout d'un certain temps
  - Inconvénients:
    - → Reconfiguration de l'anneau si nouveau site ou panne de sites
    - → Perte de jeton; il faut générer un nouveau jeton : délicat

R. CHALON MRI-RTS6 24

## Contrôle décentralisé

- Techniques sans vote:
  - Systèmes à base de verrous
  - Techniques de « passage de main » (*floor-passing*)
  - Transactions
  - Transformation d'opérations
- Techniques avec vote:
  - Consensus par majorité
  - Vote pondéré
  - Write-all-read-any
  - Vote avec témoin
  - Vote dynamique
  - Vote de classe
  - Vote multidimensionnel
  - Vote hiérarchique

R. CHALON MRI-RTS6 25

## Systèmes à base de verrous (*locks*)

- Pose de verrous en lecture/écriture sur les données:
  - Attente éventuelle si un verrou est déjà posé; éventuellement problème d'interblocage (*dead-lock*)
  - Nécessité d'une granularité suffisante pour ne pas bloquer
  - Quand poser les verrous : à la sélection des données (avec la souris par exemple) ou au moment d'accomplir l'action ?
  - Quand libérer les verrous ?
- Techniques avancées:
  - *Tickle lock* :
    - → libération automatique des verrous si l'utilisateur est inactif pendant un certain temps
  - *Probabilistic lock* :
    - → Verrou optionnel avec timeout. Si le verrou n'est pas donné avant le timeout alors l'application (ou l'utilisateur) peut s'en passer et accepter les inconsistances qui en résulteront;
    - → Les inconsistances seront résolues comme dans les méthodes optimistes

R. CHALON MRI-RTS6 26

## Technique de « passage de main » (*floor-passing*)

- *Floor-passing* explicite:
  - Un utilisateur U1 qui a le contrôle passe la main explicitement à un utilisateur U2.
- *Floor-passing* implicite avec contrôleur centralisé:
  - Un utilisateur U1 demande la main et le contrôleur lui la donne. Lorsqu'il n'en a plus besoin, il rend la main au contrôleur.
  - Si l'utilisateur garde la main trop longtemps, le contrôleur peut décider de la lui retirer pour la donner à un autre utilisateur.
- *Floor-passing* implicite avec coordination distribuée:
  - La demande de main est « *broadcastée* » à tous les utilisateurs par l'utilisateur qui veut obtenir le contrôle et l'utilisateur qui a la main la passe (éventuellement avec un délai)
  - Problème si la demande est émise en même temps qu'il y a un transfert de contrôle en cours ➔ il faut refaire la demande périodiquement !



## Transactions

- Utilisable si les transactions sont très courtes et les conflits entre opérations concurrentes résolues très vite.
- Inadapté à des données trop grosses ou des durées trop longue:
  - Probabilité de pannes augmente et il n'est plus possible de décider si la transaction a réussi ou échoué et les verrous posés ne sont pas libérés
- Facile à intégrer dans un collecticiel en utilisant une base de données (car beaucoup de bases de données fournissent un système de transaction)
- Utilisé principalement pour les collecticiels asynchrones

## Rappels sur les transactions

- Définition:
  - Unité d'accès à des données (locales ou réparties) qui doit être exécutée entièrement ou alors pas du tout (*commit* ou *abort*)
- Propriétés ACID =
  - Atomicité: de l'extérieur, la transaction apparaît comme indivisible
  - Consistance (ou cohérence): la transaction ne viole pas les invariants du système
  - Isolement: les transactions concurrentes n'interfèrent pas entre elles
  - Durabilité: lorsqu'une transaction réussie, son résultat est permanent
- Transactions réparties:
  - Nécessité d'un protocole d'engagement réparti à 2 ou 3 temps (2PC / 3PC – phases commitment)
  - Résolution des conflits :
    - → Sérialisation des transactions par la pose de verrous

R. CHALON MRI-RTS6 29

## Transformation d'opérations

- Utilisé :
  - Dans le cas d'accès à de petites données (caractères, mots, …)
  - Dans des systèmes synchrones très couplés (WYSIWIS strict)
  - Lorsque les transactions et la notion de sérialisation stricte des opérations est trop lourde
- Principe:
  - Au lieu de se baser une sérialisation stricte des opérations comme dans les transactions, on s'appuie sur des propriétés de précédences des opérations
  - Les différents sites peuvent exécuter les opérations dans une ordre différents si elles sont capables de transformer les opérations de manière à garantir la cohérence:
    - → En général $f \circ g \neq g \circ f$, mais il est possible de trouver des transformées f' de f et g' de g telles que $f' \circ g = g' \circ f$

R. CHALON MRI-RTS6 30

## Techniques de vote

- Développées pour des systèmes de fichier distribués
- Techniques intéressantes pour des granularités grossières des données
  - Car surcharge importante apportée par le processus de vote
- Peu utilisé en pratique dans les collecticiels
- Principe:
  - Pour accéder à un bloc d'informations répliquées:
    - → Un nœud demande un vote à tous les nœuds qui ont un réplica
    - → Tous les nœuds qui peuvent participer (c-à-d le réplica est accessible sans verrou) votent
    - → La somme des votes doit dépasser un quorum pour que l'accès soit obtenu.
  - Les modifications doivent être réalisées par une transaction classique pour garantir la consistance :
    - → Le vote peut être vu comme le « prepare to commit » d'un 3 PC

R. CHALON MRI-RTS6 31

5

## 5 – Middleware pour les collecticiels

- Besoin pour les services logiciels destinés aux collecticiels:
  - Support pour un couplage flexible : possibilité pour les utilisateurs de configurer eux-mêmes la granularité des interactions avec les autres utilisateurs
  - Gestion de version des données manipulées ainsi que la possibilité de fusion (d'un travail privé avec le travail de groupe)
  - Support de la conscience de groupe (*awareness*)
  - Possibilité de combiner collaboration synchrone et asynchrone
  - Moyen de transformer facilement des applications existantes non-collaboratives en applications collaboratives
  - Des canaux d'échanges multimédia (audio et vidéo)
  - Des mécanismes de *workflow* afin d'aider à la coordination de groupes

R. CHALON MRI-RTS6 32

## Caractéristiques

- Caractéristiques d'un middleware pour collecticiels:
  - Facilité d'emploi pour le programmeur :
    - → Possibilité d'abstraire les aspects distribués du collecticiel par exemple en supportant des données répliquées de manière transparente
  - Flexibilité: utilisable pour une large gamme de collecticiel ou au contraire spécialisé à un certain type
  - Performance: tant au niveau du débit et des délais sur le réseau que de la réactivité de l'interface utilisateur
  - Robustesse: faculté à résister aux pannes des machines et aux aléas du réseau
  - L'indépendance à une architecture réseau particulière (client/serveur, peer-to-peer, …)
  - La portabilité: fonctionnement dans des environnements hétérogènes: stations de travail, ordinateurs portables, PDA, voire téléphones mobiles
  - L'exigence en terme de ressources (mémoire, processeur, disque)



## Framework SMAC

- SMAC = Services fo Mobile Applications and Collaborations
  - Framework adapté au travail collaboratif capillaire
  - Collaborations synchrone et asynchrone
  - Propose un modèle d'application permettant :
    - → de s'abstraire des mécanismes sous-jacents
    - → de faciliter la communication et la coordination de composants distribués
    - → de gérer les déconnexions et les erreurs réseau
- Contraintes :
  - Architecture abstraite indépendante des architectures réseau.
  - Qualité de service ajustable.
  - Fonctionner sur dispositifs mobiles et des acteurs mobiles

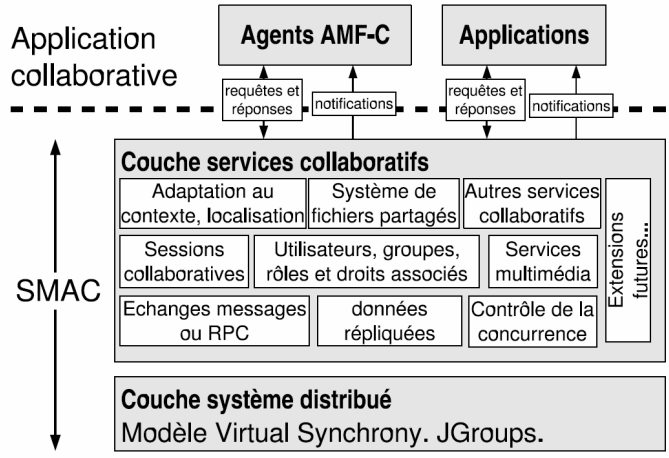
## SMAC

- Caractéristiques:
  - Communication multicast et unicast
  - Communications réseaux synchrone (type RPC) et asynchrone
  - Notification d'événements distants
  - Mécanismes de gestion de la concurrence d'accès
  - Partage transparent de données
  - Contrôle explicite de la manière dont l'application est centralisée / distribuée
  - Système de fichiers collaboratif
  - Contrôle d'accès
  - Gestion des utilisateurs, des roles et des groupes
  - Gestion simple de workflow pour les sessions collaboratives
  - Assurer la persistance des ressources collaboratives lorsque les participants sont déconnectés
  - + Outils collaboratifs de haut niveau (système de vote, télépointeur, etc.) construits sur les couches basses



## Architecture

- Modèle en 3 couches:



## Couche système distribué

- Caractéristiques :
  - L'intégrité du travail doit être garantie par rapport à la défaillance d'un des processus constituant le collecticiel ou d'un lien réseau. Cela est d'autant plus important dans des réseaux sans-fil où les déconnexions sont fréquentes.
  - Garantir la cohérence des données partagées par les différents processus. Il faut donc des techniques de réplications des données en local (pour des raisons de performances) tout en garantissant la synchronisation avec les autres processus



## Modèle Virtual Synchrony (1/3) [Birman 87]

- Système de communication de groupe
  - Systèmes distribués à tolérance de faute
- Gérer les défaillances des membres d'un groupe de communication en simulant un environnement dans lequel les échanges réseau sont fiables au sein d'un groupe et excluant les processus fautif.
- Caractéristiques :
  - GMS (Group Membership Service) : tient à jour la liste des processus actifs du groupe de communication.
  - Cette liste est broadcastée périodiquement (« views ») aux membres du groupe.
  - Ordonnancement des paquets multicast crucial, notamment les views.
  - Système complexe, mais une fois en place, simple pour les couches clientes.

## Modèle Virtual Synchrony (2/3)

- Gestion de l'ordonnancement des messages reçus et des notifications ; différentes formes d'ordonnancement :
  - *fbcast* (FIFO broadcast): les messages envoyés par un émetteur sont délivrés dans lequel ils ont été émis
  - *cbast*: ordonnancement causal. Généralisation du FIFO à des fils d'execution répartis sur plusieurs processus
  - *abcast*: ordonnancement identique de tous les paquets multicast pour tous les membres du groupe
  - *cabcast*: abcast + cbcast
  - Primitive *flush* permet de forcer la réception de tous les messages en cours par leurs destinataires. Utilisée pour les « views »
  - *dynamically-uniform multicast*: garantie de remise soit à tous les membres d'un groupe, soit à aucun en cas de défaillance. Contrainte très forte possible avec des mécanismes de transactions reparties (2PC, 3PC)



## Modèle Virtual Synchrony (3/3)

- Grâce au GMS et aux propriétés multicast, on garantit
  - que tous les membres voient la même succession de *views*
  - que tous les messages appartenant à une *view* sont délivrés à tous les membres avant de passer à une autre *view*
- *Gap-freedom*:
  - garantie en cas de défaillance d'un émetteur que:
    - → si un message m est émis par lui et reçu par d'autres processus,
    - → alors tous les messages émis auparavant par cet émetteur seront reçus par ses destinataires
- Le modèle Virtual Synchrony permet de voir le système comme un ensemble de processus qui reçoivent tous les mêmes messages dans le même ordre.

## JGroups

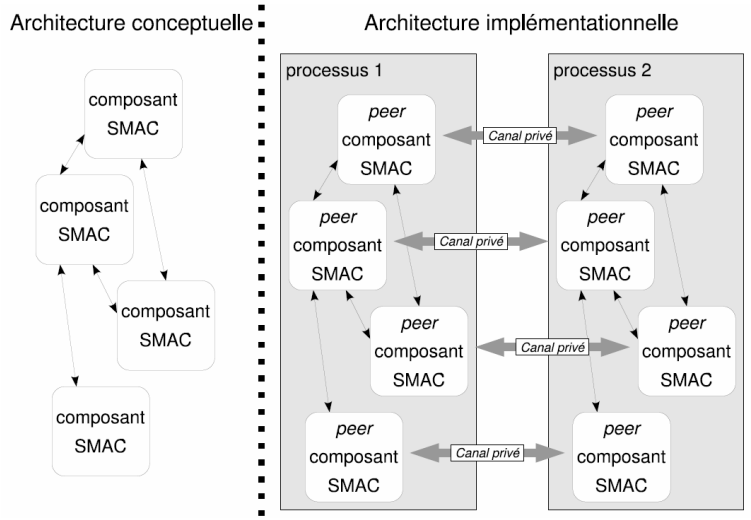
- JGroups
  - Système de communication de groupe.
  - Java, open source.
  - Utilisé pour réaliser le *clustering* du serveur d'application JBoss.
  - Permet de construire des piles de protocoles (classes java) modulaire.
    - → Abstraire le transport physique des paquets: par exemple, transport unicast TCP simulant le multicast ou transport multicast vrai UDP
    - → Moduler facilement les propriétés de la pile de protocole: JGroups supporte des piles présentant les propriétés de Virtual Synchrony
  - Mécanisme de transfert d'états lors de la connexion d'un processus à un groupe de communication. Permet d'implémenter la gestion des *latecomers* (personnes arrivant en cours de session)
  - Abstractions de haut niveau comme des structures de données répliquées (table de hashage, arbre, liste) ou des outils de vote.



## Couche Services collaboratifs

- Structurée comme un Framework
  - Notion de composant: classe Component à sémantique réduite
  - Gère pour la couche supérieure la mise en place des canaux de communication et offre les services nécessaires pour échanger des données sur ces canaux
  - Extensible
- Noyau de SMAC:
  - Gestion de la connexion réseau
  - Gestion du cycle de vie des composants: instanciation, destruction, persistance, réplication, identification (système de nommage)
  - Échange de messages entre les instances de composants
  - Postage d'événements et délivrance de ces événements aux objets abonnés à ces événements
  - Authentification des utilisateurs, contrôle d'accès

## Architectures






## Mise en œuvre (1/2)

- Classe Component:
  - Référencés par un identifiant unique
  - Lorqu'une application instance un composant, il instancie en réalisé un *peer* (instance locale du composant)
  - Lorsqu'un composant est instancié pour la 1ère fois un canal de communication privé est créé
  - Lorsqu'un un nouveau *peer* d'un composant est instancié, SMAC connecte automatiquement le canal de communication aux autres *peer* déjà existant
    - → Un état est demandé à l'un des peer existant et est communiqué au nouveau peer pour qu'il se synchronise avec l'ensemble des peer du composant
  - Les peer peuvent communiquer entre eux par les canaux privés
    - → Rien n'est imposé sur la nature des données qui transitent

## Mise en œuvre (2/2)

- Canaux de communication:
  - Permet de transférer des objets (en utilisant les mécanismes de sérialisation des objets )
  - Permettent d'envoyer des messages bloquants:
    - → Mise en œuvre de RPC (style JavaRMI) mais avec une liaison dynamique avec les méthodes distantes (*stub* non nécessaire)
- Classe SmacEvent:
  - Permet d'échanger des événements à l'ensemble des composants
    - → Alors que les canaux ne permettent que des échanges entre peer
  - Evénements typés dérivant de SmacEvent
  - Evénements générés par SMAC : SmacConnected, SmacDisconnected, ComponentCreated, ComponentDestroyed
    - → l'application peut réagir en conséquence à ces événements



## Autres fonctionnalités

- Fonctionnalités construites au-dessus du noyau:
  - Gestion des utilisateurs, des groupes, des rôles attribués aux utilisateurs
  - Gestion des collaborations (qui se situent au niveau asynchrone), des étapes au sein des collaborations, et des sessions de travail collaboratives (qui se situent au niveau synchrone)

B

## Bibliographie

- Livres :
  - Borghoff U. M. & Schlichter J. H. *Computer-Supported Cooperative Work – Introduction to Distributed Applications*, Springer 2000, ISBN 3-540-66984-1
- Rapport :
  - Imbert M. Etude d'un ensemble de services logiciels réutilisables adaptés au développement de logiciels collaboratifs en environnement mobile. Mémoire ingénieur CNAM, 2006. (http://imbert.matthieu.free.fr/cnam/memoire/memoire.pdf)