



utt
université de technologie
Troyes



Génie de Systèmes d'informations et de décisions DEA RACOR – Option RFCI

Projet de fin d'études
Avril 2002 – Septembre 2002

Extension du modèle Client Serveur pour la mobilité

Vers un modèle Java RMI mobile et dynamique

Présenté par

Houssam FAKIH

Etudiant Ingénieur DEA, UTT

fakih@essi.fr

<http://houssam.fakih.free.fr/>

Laboratoire I3S - CNRS

Equipe Rainbow

dirigé par

Michel RIVEILL

Encadrant du stage, UNICE

riveill@essi.fr

<http://www.essi.fr/~riveill>

Laboratoire I3S - CNRS

Equipe Rainbow

Remerciements

Durant l'année 2001-2002, j'ai eu le privilège de préparer mon stage de DEA RACOR au laboratoire I3S au sein de l'équipe Rainbow, mais au fil de ce stage, de nombreuses personnes y ont participé.

Je remercie donc tous les membres de l'équipe Rainbow, qui en partageant leurs savoirs, leurs expériences et leurs notes ont contribué à la réussite de mon travail et à l'exactitude de ce projet. Merci donc à :

- ✍ Michel Riveill (Professeur, UNSA) pour sa grande disponibilité, ses compétences et son encadrement.
- ✍ Mireille Blay-FORNARINO, Anne Marie PINNA-DERY et Karima BOUDAOU (Maîtres de conférences, UNSA)
- ✍ David Emsellem et Jérémy FIRESTONE (Ingénieurs de recherche, I3S)
- ✍ Olivier NANO et Pascal RAPICAULT (Doctorants, UNSA)
- ✍ Mohamed Ali JOULAK et Frédéric SAGNA (Etudiants en 3^{ième} cycle, UNSA)
- ✍ Patricia LACHAUME (assistante du projet Rainbow) pour sa gentillesse exceptionnelle.

Je tiens à remercier Dominique GAITI (Responsable du DEA RACOR, UTT) qui a supporté tous les emails que j'ai envoyés lors de la recherche de sujet de stage.

Je remercie spécialement Zahia GUESSOUM (Maître de conférences, LIP6) pour son aide et ses conseils.

Quant à ma famille, c'est toujours difficile de trouver les mots. Sans vous, ce travail sera rien. Alors je n'ai qu'à dire : Mille Mercis.

(Sommaire)

I. PROBLÉMATIQUE ET OBJECTIFS DU STAGE.....	4
II. ETAT DE L'ART.....	6
A. PARADIGME DU MODÈLE CLIENT/SERVEUR MOBILE.....	6
a) <i>Adaptation consciente de la mobilité, Mobile-Aware Adaptation</i>	6
b) <i>L'accès aux données mobiles, Mobile Data Access</i>	6
c) <i>L'extension du modèle Client/Serveur traditionnel, Extended Client-Server model</i>	6
1. Niveau de fonctionnalités	8
2. Place d'exécution du proxy	8
1.1. Le modèle client serveur avec un proxy côté réseau fixe.....	8
1.2. Le modèle client serveur avec un proxy côté réseau mobile.....	9
1.3. Le modèle client serveur intercepté.....	10
1.4. Le modèle client serveur dynamique.....	11
1.4.1. Architecture flexible du client serveur	11
III. ETUDE DE CAS.....	13
A. BAYOU.....	13
a) <i>Modèle</i>	14
b) <i>Résolution des conflits spécifique aux applications</i>	15
c) <i>Gestion des copies de bases de données</i>	17
d) <i>Applications</i>	17
• Meeting Room Scheduler	17
B. MOLÈNE.....	19
a) <i>Nature générique</i>	19
b) <i>Adaptation dans le système Molène</i>	19
c) <i>Le système Molène</i>	21
1. Les outils Molène.....	21
2. Les services Molène.....	22
C. ICECUBE.....	24
a) <i>Introduction</i>	24
1. Phase du mode non connecté.....	24
2. Phase symbolique.....	24
3. Phase de simulation.....	25
4. Phase de sélection.....	25
IV. APPROCHE MÉTHODOLOGIQUE.....	26
A. APPEL DE MÉTHODE À DISTANCE, REMOTE PROCEDURE CALL.....	26
a) <i>Présentation</i>	26
b) <i>RPC et Réseaux sans fils</i>	27

c) <i>RPC adaptée pour les réseaux mobiles</i>	27
B. JAVARMI	28
1. <i>Architecture</i>	29
2. <i>Mode Opérateur</i>	29
3. <i>Modèle Java RMI adapté aux réseaux sans fils</i>	30
1. Scénario dans le cas d'une déconnexion prévue	30
2. Scénario dans le cas d'une déconnexion brusque.....	30
4. <i>Agenda Mobile, Application</i>	30
1. Implémentation en Java RMI.....	31
1.1. Définition des interfaces	31
1.2. Diagramme de classes - Côté client	32
1.3. Diagramme de classes - Côté serveur	32
1.4. Fonctionnement en mode déconnecté.....	33
1.5. Phase de réconciliation	33
1.6. Exigences des utilisateurs	33
2. Vers une implémentation dynamique et adaptée aux réseaux sans fils	34
2.1. Outil Noah	34
2.2. Agenda Mobile et l'outil Noah	35
2.3. Exemple d'utilisation des interactions dans Agenda Mobile.....	35
V. CONCLUSION	37

I. Problématique et Objectifs du stage

Les progrès récents dans le domaine de la technologie des réseaux sans fils et des appareils portables (tels que les stations de travail, les PDAs, les téléphones, les bornes interactives, etc..) ont introduit un nouveau paradigme dans le domaine de l'informatique, il s'agit de l'informatique mobile.

Ce nouveau paradigme est caractérisé par la mobilité de ses utilisateurs qui utilisent de plus en plus souvent de multiples environnements ayant des caractéristiques très différentes qui entravent leur mobilité. Les réseaux mobiles doivent offrir une possibilité de connexion permanente quelque soit la localisation géographique à travers de multiples points autorisant la mobilité. Ils devront permettre aux utilisateurs d'accéder à des services d'information, à travers une infrastructure partagée, sans tenir compte de leurs déplacements ou de leurs localisations, etc.

De nouveaux challenges d'ordre technique sont alors introduits en ce qui concerne **le moyen d'accès aux informations**. On rappelle que les techniques traditionnelles d'accès à l'information répartie sont fondées sur des hypothèses tel que la localisation des serveurs ainsi que la connexion entre les différents acteurs ne changent pas au cours d'une session. Malheureusement dans le domaine de l'informatique mobile, ces hypothèses ne sont que rarement valides.

L'informatique mobile se distingue de son homologue classique, l'informatique avec une connexion fixe, par :

1. La mobilité

- **La mobilité des stations de travail** ce qui entraîne des ruptures de communication.
- **La mobilité des utilisateurs** qui utilisent au cours d'une même session applicative des multiples terminaux.

2. **Les contraintes** imposées par les ressources mobiles telles que la bande passante limitée des réseaux sans fils, la durée de vie de la batterie de l'appareil portable, la place mémoire et la vitesse du processeur.

La mobilité implique que les utilisateurs peuvent se connecter à partir de différents points d'accès à travers les liaisons sans fils et peuvent préférer de rester en mode connecté pendant leurs déplacements malgré les déconnexions éventuelles intermittentes. Les liaisons sans fils ne sont pas très fiables, de plus les hôtes mobiles sont alimentés par des batteries d'une durée de vie ne permettant pas un fonctionnement permanent.

Il nous semble essentiel que l'utilisateur puisse aisément s'adapter au changement de contexte en fonction de l'environnement courant. Ces possibilités constituent la base du nomadisme.

Il demande la découverte de services, l'association et la coopération de services, l'adaptation dynamique de ceux-ci, la localisation, la disponibilité d'architectures fiables et sécurisées.

Les solutions qui doivent être développées permettront de tirer parti de l'avènement de nouveaux types de réseaux et en particulier de permettre la continuité de connexion entre les différents types de réseau (réseaux locaux d'entreprise, réseaux sans-fils et connexions par modem). Les limitations et les contraintes citées ci-dessus introduisent des variations au modèle fixe de communication traditionnelle qui vont au delà de la couche réseau et affecte la couche application en elle-même.

La spécialisation du fonctionnement sera faite à l'aide des interactions qui permettent d'adapter dynamiquement le fonctionnement d'un composant et de faire communiquer "après coup" deux ou plusieurs composants logiciels déjà en fonctionnement et qui n'avaient pas été construits pour communiquer ensemble. L'outil NOAH (<http://noah.essi.fr>) développé au sein de l'équipe rainbow était utilisé pour permettre la mise en œuvre de ces mécanismes d'interaction entre les composants déjà existants.

L'objectif du mon stage de DEA est d'adapter le modèle client – serveur qui ne supporte pas les déconnexions même temporaires pour permettre un fonctionnement même dégradé de service dans les réseaux mobiles. Nous mettons en évidence les limites du modèle client – serveur traditionnel actuellement largement utilisé pour construire des applications réparties ; puis nous proposons à partir d'une étude de l'existant quelques ébauches de solution en particulier utilisable dans le cadre du modèle Java RMI. En effet, si l'on souhaite utiliser un intergiciel de type Java RMI dans le cadre de réseaux sans fils, il serait souhaitable de modifier le modèle d'appel de procédure utilisé. Mon but sera donc de faire des propositions en ce sens.

II. Etat de l'art

A. Paradigme du modèle Client/Serveur mobile

Les axes de recherche sur le modèle Client/Serveur mobile peuvent être classés en trois groupes [JING et al, 1999] qui dépendent mutuellement l'une de l'autre.

- L'adaptation consciente de la mobilité, *Mobile-Aware Adaptation*
- L'extension du modèle Client/Serveur traditionnel, *Extended Client-Server model*
- L'accès aux données mobiles, *Mobile Data Access*

a) Adaptation consciente de la mobilité, *Mobile-Aware Adaptation*

Le dynamisme des environnements mobiles et les limitations de ses ressources font de l'adaptation une technique importante à prendre en compte lors de la conception des applications et des systèmes mobiles. Le paradigme de l'adaptation consciente de la mobilité aborde les techniques et les stratégies utilisées par les systèmes ou les applications pour répondre aux changements de l'environnement, et à leurs exigences en terme de ressources.

b) L'accès aux données mobiles, *Mobile Data Access*

L'accès aux données mobiles concerne les politiques et les stratégies qu'il faut admettre pour préciser, entre autres, quel type de données il faut envoyer aux utilisateurs, comment structurer les données sur ces réseaux mobiles non filaires, et comment assurer la consistance des données dans le cache du client. Ces stratégies adaptatives dépendent largement du type des liaisons de la communication, de la connectivité des hôtes mobiles et des exigences de consistance imposées par chaque application.

c) L'extension du modèle Client/Serveur traditionnel, *Extended Client-Server model*

Une des caractéristiques importantes de ces extensions est le partage dynamique des fonctionnalités et des responsabilités entre le client et le serveur. Ce paradigme comprend plusieurs architectures qui étendent le modèle client serveur traditionnel afin de permettre un partage des fonctions entre le client et le serveur.

On rappelle d'abord la définition d'un serveur dans un système d'information. Un serveur est toute machine qui tient une ou plusieurs copies d'une base de données [JING et al., 1999]. Un client pourra accéder alors aux données existantes sur n'importe quel serveur avec lequel il est capable de communiquer.

Un système client serveur traditionnel considère que les hôtes serveur et client sont des hôtes fixes et que la connexion physique entre ces deux hôtes l'est encore. Les fonctionnalités sont alors statiquement partagées entre le client et le serveur.

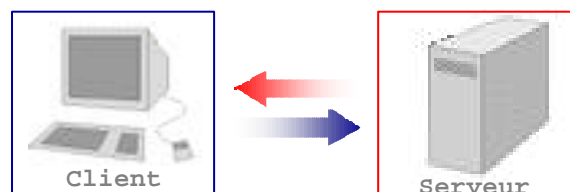


Figure 1 - Architecture Traditionnelle du modèle client - serveur

Dans ce modèle, ni le client ni le serveur ne sont conscients de la mobilité. Prenons, par exemple, le modèle client serveur reposant sur un appel de procédure à distance (voir page 28). Dans ce modèle, le client envoie une requête et reste bloqué en attendant la réponse. Si le serveur tombe en panne ou la connexion entre le client et le serveur sera perdue. Le client reste bloqué en attendant une réponse du serveur tandis que le serveur n'est pas conscient de la demande (requête) du client.

Afin de résoudre ce problème et d'étendre ce modèle pour qu'il convienne aux applications mobiles, Un processus intermédiaire est introduit entre le client et le serveur. Dans une telle architecture, on trouve que la distinction entre les fonctionnalités du serveur et celles du client peuvent être temporairement floues [Satyanarayanan, 1996], ainsi le modèle client serveur mobile pourra être présenté comme le montre la figure ci-dessous :

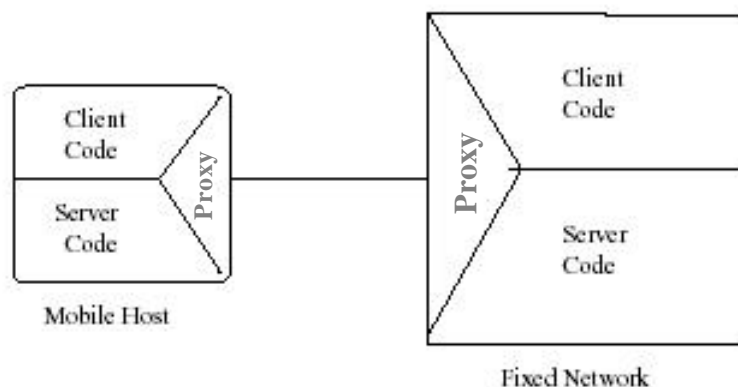


Figure 2 - Extension du modèle Client-Serveur [Jing et al., 1999]

Les ressources limitées du client peuvent éventuellement exiger à certaines opérations, normalement exécutées sur le poste client, d'être exécutées sur la machine serveur ayant toujours plus de ressources que la machine cliente. Inversement, les déconnexions fréquentes intermittentes qui caractérisent le réseau mobile obligent à l'application de remplacer éventuellement le serveur (localement sur le site client) afin de rendre transparent la déconnexion à l'utilisateur.

Les fonctionnalités proposées par ce processus intermédiaire qu'on appellera dans la suite de ce rapport 'proxy' dépendent de deux paramètres :

- Le niveau de ses fonctionnalités
- Et sa place d'exécution

1. Niveau de fonctionnalités

Il en existe trois : Le plus haut niveau propose un seul proxy pour assurer toutes les fonctionnalités des applications sur la partie cliente mobile. Le niveau moyen associe à chaque service, tel la recherche sur le Web [Fox et al., 1996] ou les bases de données [Lei et al., 1999], un proxy. Dans ce cas, l'utilisation de différents services implique l'utilisation de plusieurs proxys. Et finalement, le niveau le plus fin est celui des proxys attachés à des applications spécifiques [Zenl et al., 1996]

2. Place d'exécution du proxy

Suivant la position du proxy, on peut distinguer plusieurs extensions du modèle client serveur :

- Le modèle client serveur avec un proxy côté réseau fixe
- Le modèle client serveur avec un proxy côté réseau mobile
- Le modèle client serveur intercepté
- Le modèle client serveur dynamique

1.1. Le modèle client serveur avec un proxy côté réseau fixe

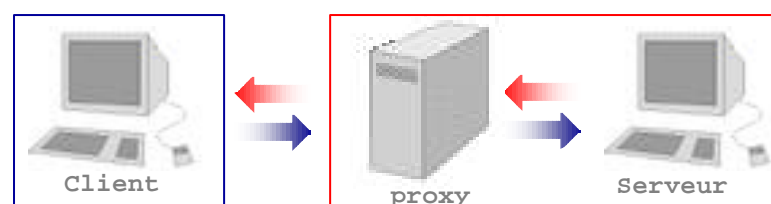


Figure 3 - Modèle Client-Serveur avec un proxy côté réseau fixe

Le proxy côté réseau fixe représente une copie du client représentant un regroupement de différentes requêtes. Il peut s'occuper de l'exécution de plusieurs opérations initialement prévues pour être exécutées sur la partie cliente. Ceci permet de :

- Réduire le code de l'application cliente,
- Réduire l'utilisation de l'énergie de la batterie puisqu'on utilise moins les ressources.

Une telle approche ne pourra pas être adoptée dans les réseaux mobiles car le problème de la non disponibilité du serveur bloque le client et ainsi il ne pourra jamais travailler en mode non connecté.

1.2. Le modèle client serveur avec un proxy côté réseau mobile



Figure 4 - Modèle Client-Serveur avec un proxy côté réseau mobile

Le proxy dans ce modèle correspond à une copie de serveur ayant une copie partielle des données. La plupart des modèles adoptant cette architecture permet au client mobile de travailler en mode non connecté. Le proxy aura alors les fonctionnalités nécessaires pour agir comme un serveur léger qui répond à toutes les requêtes du client. L'architecture du client dépend de l'application ou du service auquel le proxy est attaché. La fonctionnalité de base consiste à enregistrer localement toutes les requêtes demandées en mode non connecté puis de les retransmettre après re-connexion.

Ce modèle pourra être convenable dans le cas des connexions à faible débit. Le proxy peut traiter les données avant leur transmission depuis ou vers le réseau mobile. Et ceci pourra optimiser l'utilisation du réseau en

- Réduisant la sollicitation du réseau ce qui diminue le débit circulant sur le réseau et ceci en compressant les données [Fox et Brewer, 1996 ; Noble et al., 1997]
- Remettre en ordre les messages pour prendre en compte un traitement urgent des données prioritaires [Kistler et Satyanarayanan, 1992]

La technique de cacher des données pourra être utilisée dans le cas des connexions faibles. Le proxy cache les données souvent demandées et il est responsable d'assurer la consistance de ses données.

1.3. Le modèle client serveur intercepté

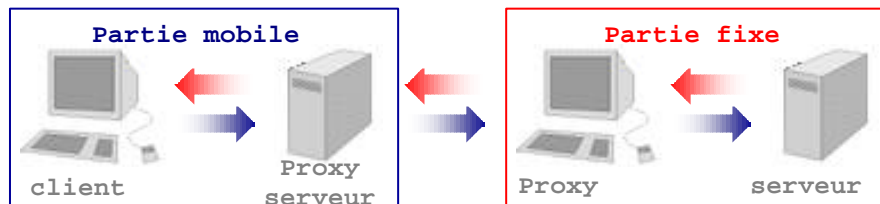


Figure 5 - Modèle Client-Serveur intercepté

Dans ce modèle, deux proxys sont installés sur le chemin des données provenant de la partie mobile et allant vers cette partie : Un est installé sur la partie mobile et l'autre sur la partie fixe.

Le proxy mobile agit comme un serveur local pour les applications sur l'hôte mobile tandis que le proxy fixe agit comme un client local sur le réseau fixe. La coopération entre ces deux éléments doit permettre d'améliorer les moyens afin de réduire la transmission des données sur le réseau non filaire. Elle doit également assurer la disponibilité des services et cacher toute déconnexion et la rendre transparente pour l'utilisateur.

Le mode déconnecté est facilement assuré par ce modèle. Les données fréquemment demandées sont cachées sur le poste du client [Housel et al., 1998]. En plus, les requêtes qui ne peuvent pas aboutir seront enregistrées dans une file d'attente sur le réseau mobile en attendant la reconnexion pour les retransmettre. De la même façon, des réponses des requêtes qui ne peuvent pas être envoyées à l'utilisateur seront enregistrées sur la partie fixe pour qu'elles soient retransmises une fois que le client sera reconnecté.

Le support d'une connexion faible sera assuré par une compression des données ou un filtrage de ces données comme on l'a montré dans le modèle précédent.

On voit alors que le modèle intercepté sépare clairement les responsabilités entre le proxy sur la partie fixe et celui sur la partie mobile. Il permet, simultanément, une grande collaboration entre les deux parties. Les deux proxys évitent alors aux applications d'inclure

dans leur code la gestion de toutes les opérations non fonctionnelles reliées à la gestion du réseau non filaire.

1.4. Le modèle client serveur dynamique

L'extension des applications déjà existantes dans les modèles précédents adopte une séparation statique des fonctionnalités entre la partie fixe et la partie mobile. D'où les systèmes adoptant cette approche n'ont pas la flexibilité nécessaire pour s'adapter aux variations caractérisant les réseaux non filaires. Ce modèle représente une extension du modèle intercepté avec la flexibilité nécessaire aux réseaux non filaires pour lui permettre de séparer dynamiquement la charge du travail entre les deux parties fixe et mobile et ceci suivant les variations des caractéristiques de l'environnement et des ressources disponibles. Pour permettre à l'utilisateur de travailler en mode non connecté, Les fonctionnalités du proxy du côté mobile pourront être étendues et déplacées depuis ou vers le proxy mobile.

Le modèle dynamique reprend les avantages du modèle intercepté en ce qui concerne la séparation entre le code de l'application et le code des ses propriétés non fonctionnelles. Cependant, la redistribution des fonctionnalités demeure une tâche délicate pour les développeurs de l'application.

Dans ce modèle, le client a une architecture dite flexible.

1.4.1. Architecture flexible du client serveur

Cette architecture généralise les deux architectures précédentes. Le partage des fonctionnalités entre le client et le serveur changent et s'adaptent d'une façon dynamique (voir figure ci-dessous)

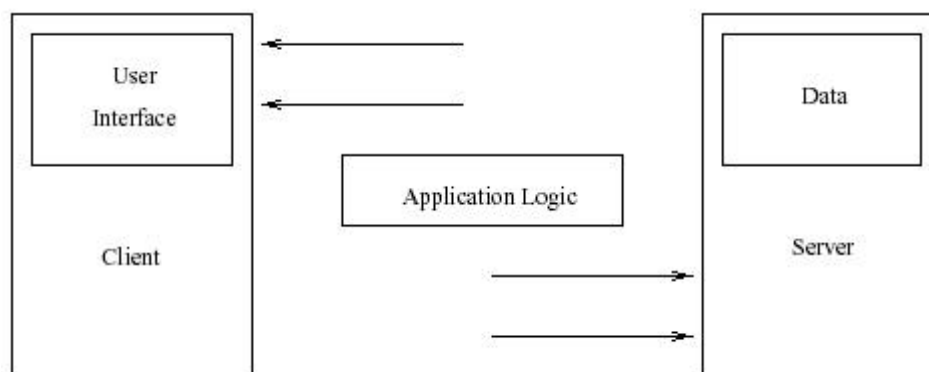


Figure 6 - Architecture Client Serveur flexible

Dans cette architecture, la distinction entre le client et le serveur peut devenir temporairement floue pour des raisons de performance et de disponibilité. En outre, la connexion entre le client et le serveur peut être établie d'une façon dynamique durant l'exécution de l'application.

Afin de récapituler, on peut reprendre la classification faite par Sumi Helal [Helal, ? ? ?] en ce qui concerne l'évolution du modèle client serveur Helal classifie cette évolution suivant la conscience de la côté fixe ou de la côté mobile par la mobilité. Ainsi, il présente les différents modèles classés en 4 groupes comme suit :

- System-transparent, application-transparent
 - Le modèle conventionnel Client/Serveur
- System-aware, application-transparent
 - Le modèle non connecté
 - Le modèle client/proxy/serveur
- System-transparent, application-aware
 - Le modèle client/serveur dynamique
 - Le modèle des agents mobiles
- System-aware, application aware

Certains auteurs considèrent que le modèle des agents mobiles appartient à la dernière famille où le client et le serveur sont conscients de la mobilité. Je rappelle brièvement le principe de fonctionnement des agents mobiles : ce sont des entités logicielles qui parcourent librement le réseau. Ces agents permettent aux utilisateurs d'exécuter les fonctions non seulement sur leurs propres hôtes mobiles mais aussi bien sur les hôtes serveurs fixes. En plus, ils permettent aux clients de télécharger du code serveur sur leurs propres hôtes pour y être exécuté. Ces agents mobiles peuvent contenir des threads, ils peuvent donc être actifs. Ils peuvent maintenir des informations sur l'état de leur environnement et prendre des décisions adaptées en les utilisant. Sur réception d'une requête, l'agent mobile se déplace vers le réseau fixe afin de communiquer localement, en tant que client, avec le serveur. Une fois le client est connecté, l'agent se déplace vers lui pour lui communiquer la réponse de la requête.

III. Etude de cas

On présente dans ce qui suit trois systèmes adaptés pour les réseaux non filaires. Ces 3 systèmes connus sous les noms : Bayou, Molène et IceCube nous permettent de voir comment les nouvelles notions du paradigme de l'informatique mobile déjà introduites ont été mises en œuvre.

A. Bayou

Bayou est un projet de recherche du laboratoire Xerox Parc. Il est conçu pour des applications collaboratives dans un environnement mobile. Dans cet environnement mobile, on trouve des ordinateurs portables qui ont des connexions intermittentes avec le réseau [Demers et al., 1994 ; Terry et al., 1994, 1995]

Le système Bayou se repose sur un modèle d'accès de n'importe quelle place, Access Anywhere Model. Il permet une réplique dynamique des données partagées. Les utilisateurs mobiles peuvent accéder librement à l'écriture et à la lecture des données partagées dans les applications ce qui peut entraîner une divergence entre les différentes copies. L'utilisateur n'a pas à se préoccuper de la propagation des mises à jour ou de la gestion des conflits. Le système bayou propage les différentes actions via le protocole de réconciliation 'Paire-Wise Reconciliation Protocol'. Bayou supporte des mécanismes spécifiques aux applications qui détectent les conflits éventuels entre les différentes mises à jour, assurent une consistance éventuelle des différentes copies du serveur, et définissent un protocole spécifique à la résolution des conflits détectés. La méthode utilisée pour la détection des conflits entre les différentes mises à jour est nommée '**Dependency Check**' cependant que la procédure utilisée pour la résolution de ces conflits est nommée '**Merge Procedure**'. L'utilisateur s'occupe de l'écriture de ces méthodes et de ces procédures qui dépendent de l'application et des préférences des utilisateurs.

Pour garantir une consistance éventuelle, Bayou est capable d'annuler les effets de certaines mises à jour déjà effectuées sur la copie des bases de données, ainsi on peut restaurer les états précédents selon un ordre global de sérialisation. En outre, Bayou permet à ses utilisateurs de

suivre le traitement de toutes les requêtes d'écriture reçues par le serveur, y inclus les tentatives d'écriture avec des conflits qui ne sont pas encore résolus.

a) Modèle

Dans le système Bayou, une copie de la base de données se trouve sur chaque serveur, les applications interagissent avec cette base de données à travers une interface de programmation Bayou API, Application Interface Programming qui est implémentée en tant qu'un stub côté client. Cette API permet à l'utilisateur d'exécuter deux types d'opérations sur la base de données : il s'agit des opérations de lecture et d'écriture. Les opérations de lecture consistent à obtenir des réponses de la base de données sur des requêtes soumises par l'utilisateur tandis que les opérations d'écriture permettent d'insérer, de modifier, et de supprimer certaines entrées de la base de données.

La figure ci-dessous présente certains composants de l'architecture Bayou. On remarque que le client et le serveur peuvent cohabiter dans le même hôte, ce qui permet à un utilisateur ayant un ordinateur portable ou un PDA de travailler en mode non connecté.

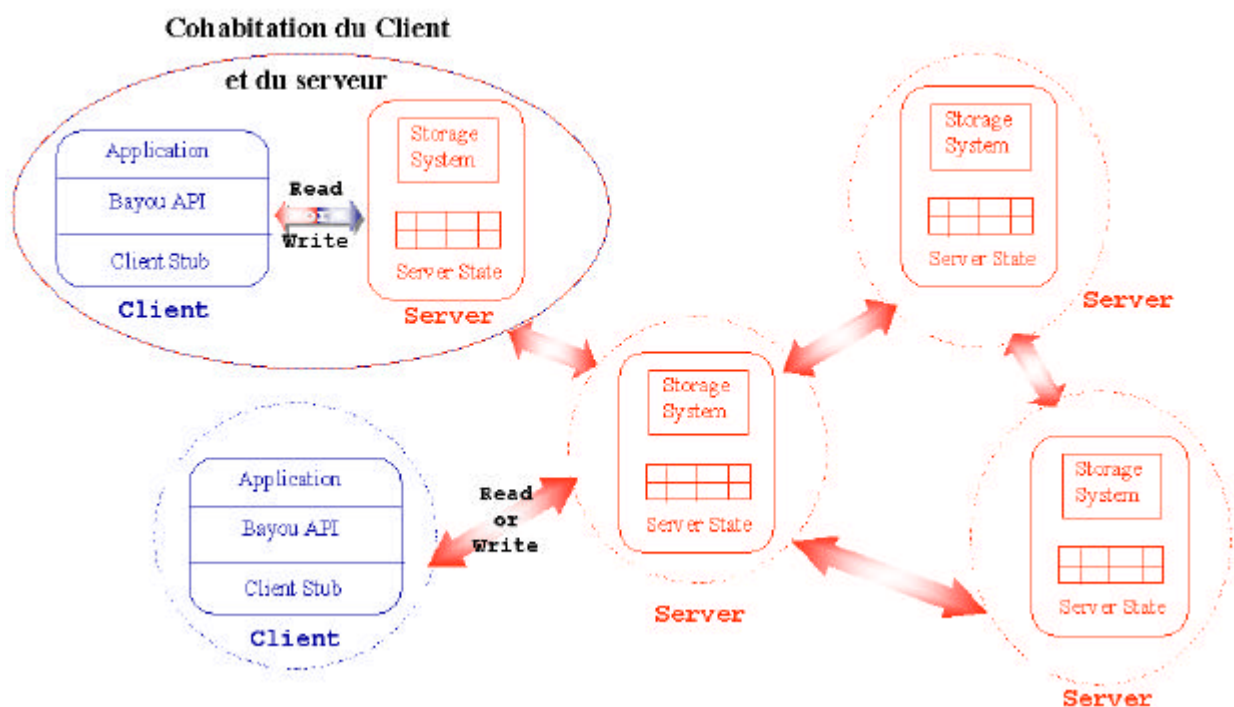


Figure 6 - Les différents composants du modèle Bayou

L'accès du client à un seul serveur est suffisant pour qu'il puisse travailler. Le client pourra ainsi lire les données portées par ce serveur et y soumettre toutes les écritures et les mises à jour éventuelles. Une fois, l'écriture ou la mise à jour sera enregistrée par le serveur, le client n'a pas à se préoccuper de la propagation de la mise à jour aux autres serveurs. En résumé, Bayou présente un modèle à réplication faible avec un accès libre en lecture et en écriture. Il assure une certaine convergence vers une consistance globale entre les différentes copies distribuées des serveurs.

b) Résolution des conflits spécifique aux applications

Bayou adopte un système de détection des conflits spécifique aux applications. Cette approche est justifiée par le fait que chaque application a ses propres définitions des conflits qui peuvent eux même varier d'un utilisateur à un autre. Ainsi, Bayou assure les moyens pour que chaque application spécifie ses propres définitions des conflits et la politique pour résoudre ces conflits. En échange, le système implémente les mécanismes nécessaires pour une détection fiable des conflits, tels qu'ils sont définis par l'application, et pour une résolution automatique des ces conflits.

La détection et/ou la résolution des conflits repose sur deux mécanismes génériques : Dependency check et merge procedure, indépendants de l'application. Ces mécanismes permettent aux utilisateurs de préciser, pour chaque opération d'écriture, comment le système doit détecter si la mise à jour résultante de cette opération pourra être en conflit avec la base de données et dans ce cas quelles sont les étapes à suivre pour résoudre ce conflit tout en respectant la sémantique de l'application.

Exprimons le fonctionnement de ces deux mécanismes avec plus de détails : On trouve que chaque opération d'écriture doit inclure un 'Dependency Check', qui permet de détecter les conflits éventuels. Reçu par un serveur, la procédure 'Dependency Check' sera exécutée sur sa base de données. Cette procédure présente un préalable pour accomplir la mise à jour incluse dans l'opération de lecture. Si un conflit est détecté, la mise à jour n'aboutit pas et le système exécute la procédure pour résoudre le conflit constaté.

Prenons un exemple : considérons une application qui permet à des utilisateurs de réserver une salle de travail pour un intervalle du temps précisé par l'utilisateur. Une opération d'écriture Bayou consiste à effectuer cette réservation. Elle peut comprendre une

procédure de Dependency Check qui consiste en une requête qui doit retourner toute réservation déjà faite et chevauchant avec la réservation souhaitée par l'utilisateur. L'opération d'écriture contiendra également la réponse prévue à cette requête.

La procédure Dependency Check peut également détecter des conflits entre deux écritures simultanées, i.e. lorsque deux utilisateurs mettent à jour la même entrée dans une base de donnée sans que l'un arrive à voir la mise à jour faite par l'autre. Des tels conflits seront détectés en enregistrant les dernières valeurs des différentes entrées mises en jeu et de vérifier les valeurs de ces entrées lors de la soumission de l'écriture.

On note que ces mécanismes sont beaucoup plus utiles que les mécanismes traditionnels, considérons pour démontrer l'utilité un exemple de virement bancaire d'une somme de 20 € entre deux comptes A et B. L'application, lit d'abord le solde du compte A et trouve qu'il est créditeur d'une somme de 115 €. Les contrôles traditionnels des accès concurrents doivent vérifier la disponibilité de la somme de 115 € sur le compte A tant que la soumission de virement n'est pas encore terminée, sinon un conflit sera détecté. La procédure de Dependency Check peut par exemple exiger qu'une somme au moins égale à 20 € soit disponible pour soumettre l'opération. Un conflit sera détecté si une autre opération rend le compte créditeur d'un solde inférieur à 20 €

Une fois, le conflit détecté, la procédure 'Merge procedure' sera exécutée sur le serveur Bayou pour résoudre le problème. Ces procédures sont des programmes écrits dans un langage interprété. Ils peuvent contenir du code et des données (relatifs à la sémantique de l'application et à la mise à jour attendue) et peuvent accéder aux données concernant l'état actuel du serveur. Chaque procédure associée à une opération d'écriture sera responsable de résoudre tout conflit détecté par le dependency check et d'appliquer la mise à jour résolue. Ce processus complet de détection des conflits, de résolution et d'exécution des opérations éventuellement corrigées sera exécuté sur chaque serveur bayou comme étant une seule opération d'écriture atomique.

Les procédures de résolution de conflits sont écrites par les programmeurs d'application. Les utilisateurs de l'application n'ont pas à se préoccuper de ces procédures de détection et de résolution des conflits. Les conflits résolus sont cachés aux utilisateurs qui ne savent que ceux n'ayant pas pu être résolus. Dans ce la procédure de résolution de conflit demeure active pour l'achèvement de l'opération d'écriture, la mise à jour révisée notera alors

le conflit détecté d'une façon qui permettra l'intervention de l'utilisateur pour résoudre le problème.

c) Gestion des copies de bases de données

Etant donnée que les copies des bases de données hébergées chez deux serveurs peuvent diverger suite à la réception des opérations d'écriture différentes, le système Bayou a une propriété fondamentale qui consiste à diffuser les mises à jour aux différentes copies de bases de données pour garantir une consistance éventuelle pour toutes les copies gérés par le système. Cependant, Bayou ne peut pas imposer des limitations strictes aux délais de propagation des mises à jour puisque ceux-ci dépendent des caractéristiques de la connexion au réseau, facteurs que Bayou ne peut pas contrôler. Afin de garantir une cohérence de toutes les copies du serveur, le modèle Bayou présente deux caractéristiques importantes : la première consiste à exécuter toutes les opérations d'écriture dans le même ordre sur tous les serveurs. La deuxième est que les procédures de détection et de résolution de sont déterministes ce qui entraîne une relation de bijection entre le conflit détecté et la résolution choisie. Ainsi, pour chaque conflit détecté, on a une et une seule procédure de résolution exécutée.

d) Applications

Le système Bayou est conçu pour supporter une variété d'applications collaboratrices réelles, telles les courriers électroniques, les agendas partagés, les bases de données bibliographiques, etc..

On présente dans ce qui suit une des applications du système Bayou, 'Meeting Room Scheduler'

- **Meeting Room Scheduler**

Cette application permet aux utilisateurs de réserver une salle de travail. Un utilisateur ou un groupe des utilisateurs peuvent réserver une salle pour un intervalle du temps donné à condition que la salle soit libre.

L'application aide les utilisateurs à faire la réservation. On suppose alors que les utilisateurs se sont mis d'accord sur la salle et sur la date de leur réunion, ils ont également choisi d'autres dates alternatives.

Les utilisateurs interagissent les uns avec les autres à travers une interface graphique qui montre le planning de réservation de la salle choisie. Les données sur cette interface sont rafraîchies périodiquement au fur et à mesure de la variation du planning. Ainsi, les

utilisateurs peuvent observer les nouveaux champs de réservation ajoutés par d'autres utilisateurs. Dans le cas d'un utilisateur en mode non connecté, une copie locale du planning de la salle est affichée. Le planning ne sera pas à jour, seules les réservations déjà acceptées avant la déconnexion seront sur le planning.

Pour réserver une salle, l'utilisateur doit choisir un intervalle du temps et remplir un formulaire décrivant la réunion. Etant donné qu'un utilisateur en mode non connecté a son affichage 'out of date', il se peut qu'il essaye de réserver une salle pour une date déjà choisie par un autre utilisateur. Les utilisateurs, qui sont bien au courant de cette possibilité, peuvent choisir plusieurs dates de réunions au lieu d'une seule date.

Une réservation pourra être immédiatement confirmé ou rejeté. Elle pourra également considérer comme 'provisoire' pour un certain temps. Les réservations provisoires sont affichées en gris sur l'écran de l'utilisateur.

B. Molène

a) Nature générique

Molène est un espace de travail qui adopte le concept orienté objet, Il s'intéresse particulièrement à la réutilisation des composants qui facilite la mise en œuvre des mécanismes de l'adaptation. L'architecture de Molène est à deux niveaux, comme la montre la figure ci-dessous :

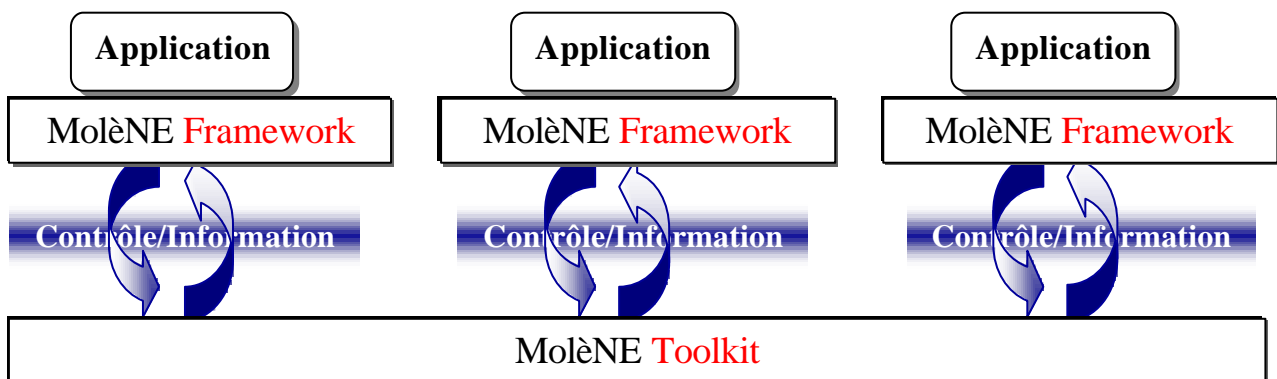


Figure 7 - Architecture à 2 niveaux de système Molène

- **Les outils génériques, Molène Toolkit**

Les outils Molène sont des outils génériques complètement indépendants de toutes applications, ils correspondent à des classes concrètes qui ne dépendent d'aucune classe abstraite. Des services tels la gestion du trafic allant de ou partant vers l'interface réseau ou la mesure de la disponibilité des ressources sont des bons candidats à être des outils génériques du Molène.

- **Les services Molène, Molène Framework**

Les services Molène sont des espaces de travail qui contiennent des fonctions spécifiques qui dépendent de l'application. Ils correspondent à des classes abstraites et sont implémentés par des classes concrètes. Toute application utilise ces services pour gérer tous les aspects liés à la mobilité.

b) Adaptation dans le système Molène

Les outils Molène offrent les moyens de contrôle de l'environnement et permettant l'adaptation de l'application à ses changements. Etant donné que le niveau Molène TOOLKIT présente une vue globale de l'état du système, les décisions d'adaptation sont prises à ce niveau suivant les exigences en terme de ressources utilisées ou demandées par toutes les applications et non pas d'une seule, ainsi que l'ensemble des ressources disponibles.

Les services Molène sont, par contre, mis en œuvre à l'aide de trois types d'objets :

- Objet Service
- Objet Micro - Service
- Objet d'implémentation

Un service Molène est représenté par un objet service dont le rôle est divisé en sous tâches encapsulées dans des objets micro – services. L'implémentation d'un micro – service particulier sera faite à l'aide d'un objet d'implémentation. La relation entre ces 3 objets est représentée par le patron de conception ci-dessous :

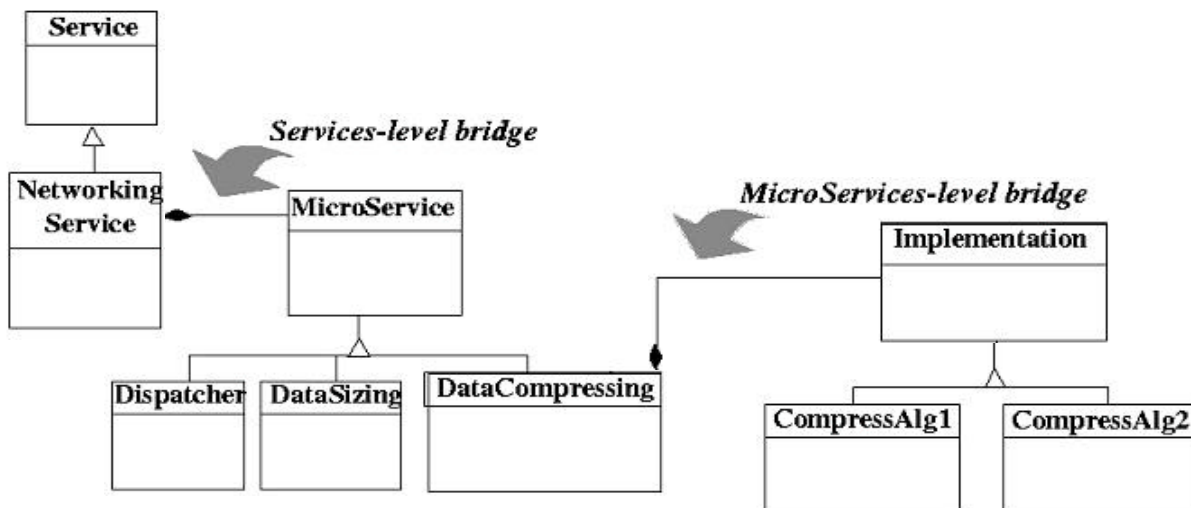


Figure 8 – Architecture à deux niveaux pour le service réseau Molène

Prenons comme exemple la gestion du trafic d'une application sur le réseau. Cette gestion est accomplie par le service Molène du réseau, Molène *Networking Service*. Comme le montre la figure ci-dessus, le *Networking Service* comporte trois micro – services, le *dispatcher*, en charge de faire suivre les données reçues du réseau aux services correspondants suivant le type de l'information, par exemple, une réponse à une requête de cache des données doit être fait suivre au service qui gère le contenu du cache ; le *Data Sizing*, responsable de réduire proportionnellement les dimensions des composants graphiques pour qu'elles soient correctement affichées sur les écrans des PDAs et des ordinateurs portables ; le *Data Compressing*, utilisé pour organiser les données et les formater. Ce service pourra être mis à jour en utilisant deux classes d'implémentation. Chacune de ces classes utilise un algorithme de compression. Chaque algorithme consomme les ressources d'une manière différente de l'autre. Le choix entre ces deux algorithmes dépend des ressources disponibles. Dans le cas

d'une batterie à faible charge, par exemple, l'algorithme qui consomme le moins de ressources sera utilisé.

c) Le système Molène

On présente dans ce qui suit les différentes fonctionnalités des outils et des services Molène. La figure ci-dessous représente quelques unes de ces fonctionnalités. La figure 10 représente en plus les liaisons qui existent entre les différents outils et services. En effet, il existe deux types de liaisons :

- Les liaisons de données, *Data Relation* : correspondent à une coopération entre les outils et les services pour contrôler les données ;
- Les liaisons de contrôle, *Control Relation* : correspondent aux opérations destinées à configurer les services Molène ou aux opérations considérées comme des méta - opérations.

1. Les outils Molène

L'administrateur système choisit les outils à exécuter dans un environnement particulier selon les fonctionnalités qu'il souhaite mettre en œuvre.

- L'outil de communication, *Communication Tool*, est responsable de l'envoi et de la réception des informations depuis ou vers l'interface réseau. Il gère également tous les aspects de la communication de l'application.
- L'outil de sécurité, *Migration Tool*, vérifie si l'exécution d'un code étranger provenant d'un site distant est permise ou non. Il fait appel à l'outil de la communication dans le cas d'un environnement distribué. Il est également utilisé par les services Molène lorsqu'ils ont besoin de changer la location d'un micro - service.
- Le *Resources Monitor*, mesure les ressources disponibles sur le hôte tel la vitesse du processeur, la capacité de la mémoire et la durée de vie de batterie. Il permet également de diffuser ces informations aux autres composants s'ils les demandent.
- Le *Applications Registry*, maintient toutes les informations concernant les applications en cours d'exécution.

- L'outil d'adaptation, *Adaptability tool*, est le cœur de l'adaptation dynamique dans le système Molène. Il déclenche l'adaptation suivant les besoins des applications et des ressources disponibles.

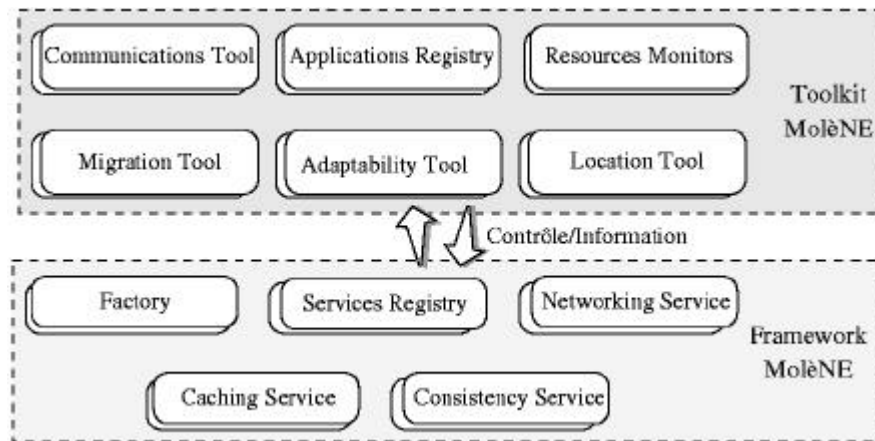


Figure 9 - Certains Outils et Services dans le système Molène

Une application initialise, tout au début de son exécution, un processus destiné, entre autre, à envoyer des informations concernant l'application et sa stratégie d'adaptation à l'adaptateur global. Une application a une clé unique (généralement un nom simple précisé par le programmeur d'application) et une priorité (généralement précisée par l'utilisateur de l'application). L'adaptateur global utilise ces informations pour donner la main aux applications correspondantes dans le cas d'un conflit.

La stratégie d'adaptation est fournie à l'adaptateur global et à l'outil de détection et de notification sous la forme d'une suite de conditions booléennes sur les ressources existantes. L'outil de détection et de notification s'occupe de détecter les variations et d'informer l'adaptateur global de toute variation. Suivant les données reçues, l'adaptateur décide si l'application doit s'adapter ou non. Si oui, il en informe l'intergiciel en précisant les conditions de l'adaptation et l'identité de l'application.

2. Les services Molène

Les services Molène représentent une interface entre les outils Molène et les applications. Tous les services sont symétriques, i.e. tout service a deux aspects : un aspect côté client et un aspect côté serveur qui fonctionnent ensemble pour effectuer leurs tâches. Une architecture client – serveur, transparente pour les utilisateurs, est alors adoptée pour mettre en œuvre cette symétrie.

Détaillons le service Molène de cache des données, Caching Service :

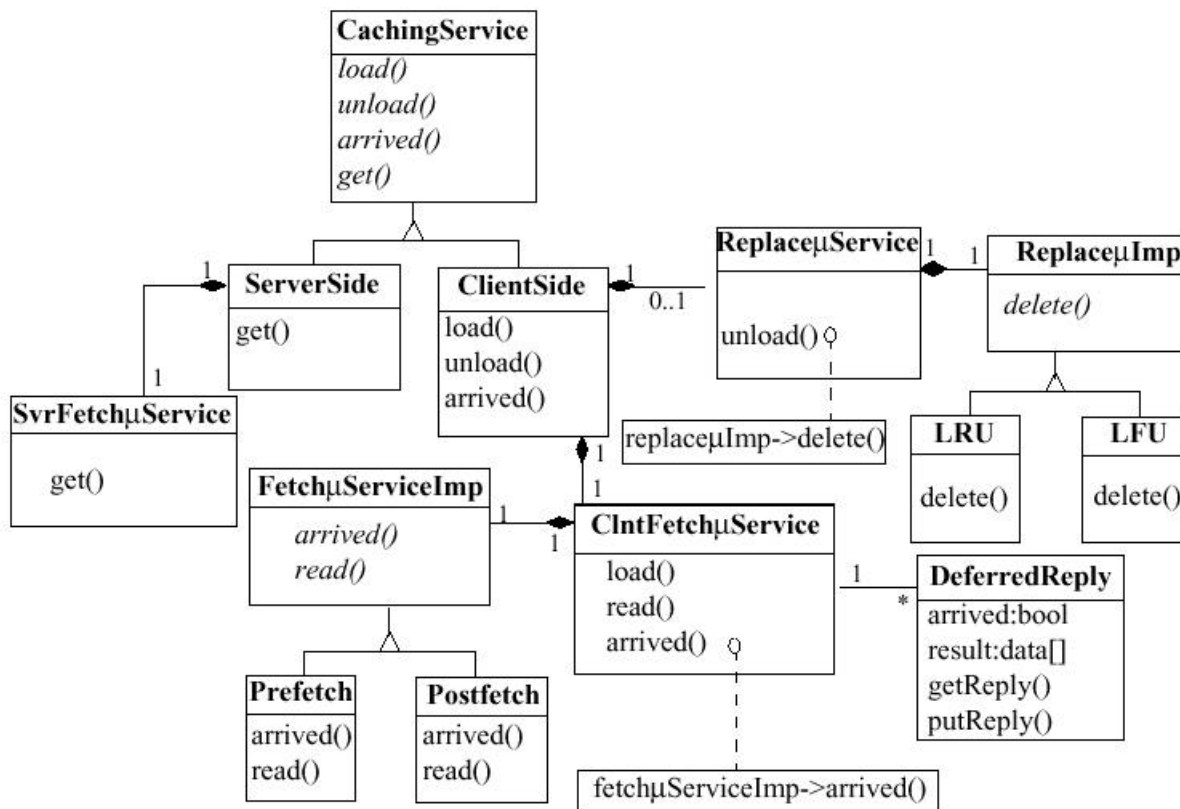


Figure 10 - Structure du service de cache dans le système Molène

La figure ci-dessus représente les différentes classes du service de cache.

Le mécanisme de cache implémente quatre méthodes : *load()*, *unload()*, *arrived()* et *get()*. La méthode *read()* utilise la méthode *load()* si les données demandées ne sont pas disponibles dans le cache de l'utilisateur. Cette dernière méthode crée un objet *DeferredReply* pour éviter de mettre l'application en attente.

La méthode *get()* est appelée sur le côté serveur lorsqu'une requête de cache des données est reçue par le service de cache. Lorsque la partie cliente reçoit la réponse, la méthode *arrived()* fait suivre les données reçues à l'objet *DeferredReply*.

La méthode *unload()* est utilisée pour supprimer des données du cache. Plusieurs stratégies peuvent être adoptées : la suppression pourra être faite, par exemple, suivant le temps de chargement des données dans le cache.

L'utilisateur peut surcharger les différentes méthodes pour adapter le comportement du système à ses propres préférences et ses propres choix.

C. IceCube

IceCube est un projet de recherche du laboratoire Microsoft, Cambridge (Grande Bretagne).

a) Introduction

IceCube est conçu pour réconcilier les données divergentes de deux ou plusieurs copies de base de données.

Ce système pourra être utilisé dans le domaine de l'informatique mobile. L'utilisateur, qui travaille en mode non connecté, met à jour les données sur sa machine. Les données sur le serveur varient également au cours du temps. Une fois reconnecté, il est probable que les données sur la machine portable soient en conflit avec les données sur le serveur. La réconciliation doit alors permettre de détecter et de résoudre les conflits pour garantir une valeur consistante des données. La définition d'un conflit et de la procédure de sa réconciliation dépend de la sémantique de l'application et des intentions de l'utilisateur.

La gestion des données partagées se fait en plusieurs phases :

- Phase du mode non connecté
- Phase symbolique
- Phase de simulation
- Phase de sélection

1. Phase du mode non connecté

L'utilisateur, en mode non connecté, accède librement en lecture et en écriture à une copie locale d'une base de données partagée. Toutes les actions de l'utilisateur sur cette copie locale de base de données sont enregistrées dans un fichier de log suivant une structure de graphe. Chaque nœud de ce graphe représente une opération sur la base de données.

2. Phase symbolique

Une fois reconnecté, les deux dispositifs se mettent en communication, ils échangent alors leurs fichiers de log. Un nouveau log sera alors généré à partir de deux fichiers échangés. Ce nouveau fichier log organise l'ordre des actions sur les données partagées en prenant en compte les différentes contraintes imposées par chaque objet. Une ou plusieurs possibilités pour la mise en ordre des différentes actions peuvent exister. Il est aussi probable que les deux fichiers soient en conflits qui ne sont pas résolubles automatiquement. Dans ce cas, le

système permet à l'application (ou l'utilisateur) d'éditer les fichiers de log afin de supprimer les actions qui introduisent le conflit.

3. Phase de simulation

Dans cette phase, on simule les résultats possibles de l'application des différents ordres trouvés pendant la phase symbolique.

4. Phase de sélection

L'application (l'utilisateur) choisit un ordre parmi les différents ordres en fonction des résultats de la phase de sélection.

Toute action sur les objets partagés est composée d'une :

- Précondition : équivalente à la procédure '*Dependency check*' de l'action Bayou. (voir page 14). Elle permet de détecter les conflits éventuels lors de l'exécution de l'opération.
- Opération : représente un sous programme accédant aux objets partagés.
- Post-condition : équivalente à la procédure '*Merge Procedure*' de l'action Bayou. (voir page 14). Elle est exécutée si un conflit détecté n'est pas résolu par les politiques de réconciliation prédéfinies.

Les préalables et les post-conditions sont exprimés dans un langage de logique du premier ordre.

IV. Approche méthodologique

Durant mon stage, je me suis intéressé à étendre dynamiquement le modèle client – serveur pour l’adapter lorsque c’est nécessaire au fonctionnement d’un réseau sans fils en fonction des besoins applicatifs. Mon challenge est de faciliter le développement des applications utilisant des terminaux nomades (assistants personnels ou autres) et des réseaux sans fils.

A. Appel de méthode à distance, Remote Procedure Call

a) Présentation

Les modèles client – serveur traditionnels sont basés sur un appel de procédure distante, RPC – Remote Procedure Call. Un modèle d’appel de procédure à distance permet à deux entités hébergées dans des espaces virtuels différents de communiquer l’un avec l’autre. L’entité cliente appelle la méthode comme si elle était une méthode locale. Cet appel

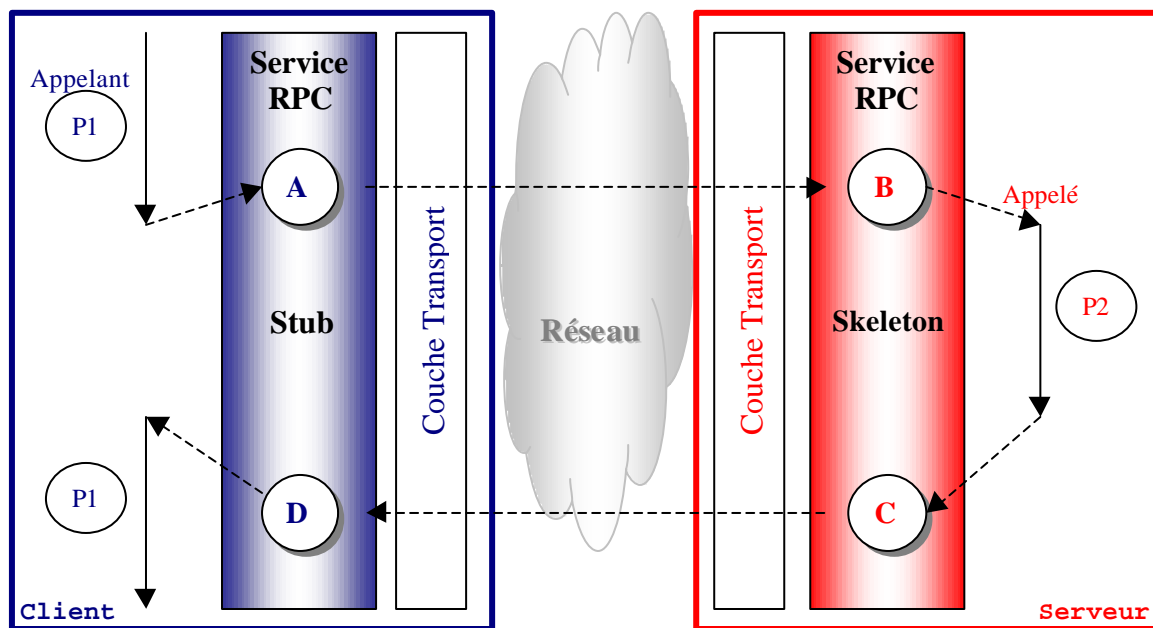


Figure 11 - Principe de fonctionnement d'un appel distant [Birrel et Nelson, 1984]

déclenche sur l'entité serveuse l'exécution de la procédure définissant la méthode. La figure ci-dessus schématise le principe de fonctionnement d'un appel distant.

Le stub représente la procédure d'interface du site client qui reçoit l'appel en mode local, le transforme en appel distant et l'envoie sous forme d'un message. Le skeleton est la procédure

sur le site serveur qui reçoit l'appel sous forme d'un message. Il fait réaliser l'exécution sur le site serveur par la procédure correspondante au message reçu (choix de la procédure) puis il envoie le résultat que le stub recevrait. Ce dernier retourne le paramètre résultat comme dans un retour de procédure.

On ne détaille pas ici les avantages et les inconvénients de cette méthode d'appel de procédure à distance. Ce qui nous intéresse est de noter que la connexion entre le client et le serveur doit être maintenue tout au long de la session faute de quoi les requêtes seront perdues : le client risque de ne pas avoir une réponse pour la requête soumise soit parce que le serveur ne l'a pas reçue, soit elle était bien reçue par le serveur mais la connexion était interrompue après l'envoi de la réponse.

b) RPC et Réseaux sans fils

Dans un environnement mobile caractérisé par des déconnexions fréquentes temporaires dues à la nature des réseaux sans fils. Cette méthode d'appel de procédure ne pourra pas être appliquée telle qu'elle est présentée. Les déconnexions fréquentes rendent impossible la connexion permanente entre le client et le serveur. Il s'avère indispensable que les deux entités en soient conscientes et prévoient un mécanisme permettant de tolérer les périodes de déconnexions tout en rendant ça transparent au client final (utilisateur de l'application). Une des solutions consiste à ce que le client va temporairement sur le site du serveur ou inversement que le serveur vient temporairement sur le site du client.

c) RPC adaptée pour les réseaux mobiles

La solution que nous envisagerons consiste à ce que le client s'adresse à un objet local sur la machine cliente et y délègue l'exécution des différentes méthodes. Cet objet que l'on appelle, `ConnectionManager`, acquitte au client la bonne réception de ses appels puis il s'occupe de les faire suivre au serveur distant s'il est disponible ou à une copie locale du serveur sinon.

Le serveur local peut être une copie complète ou une copie partielle du serveur distant. La stratégie par défaut consiste à copier tout le serveur sur la machine du client. L'utilisateur a toujours le choix de la changer en sélectionnant une partie des données pour qu'elle soit téléchargée sur sa machine.

Toute opération d'écriture sur ce serveur pourra être enregistrée dans un fichier de log puis elle sera retransmise au serveur distant une fois que la connexion se rétablirait du nouveau.

On distingue deux types de déconnexions :

- La déconnexion involontaire non prévue par l'utilisateur : Cette déconnexion involontaire pourra avoir lieu suite à une charge de batterie insuffisante, une indisponibilité du réseau, etc.. Dans ce cas, le système s'occupe de rétablir la connexion.
- La déconnexion volontaire prévue par l'utilisateur : Dans ce cas, l'utilisateur prévoit de se déconnecter du réseau pour éviter un coût de communication, ou décide de se déconnecter pour réserver les ressources de sa machine. Dans ce cas, la reconnexion au serveur distant ne sera faite que suite à une demande explicite de l'utilisateur.

Notre challenge est que cette déconnexion soit transparente à l'utilisateur dans le sens où dès que l'utilisateur ne soit plus connecté toutes les requêtes soumises au serveur seraient prises en compte et servis localement par une copie du serveur.

La reconnexion au serveur implique une retransmission de toutes les actions soumises localement en mode non connecté. L'application de ces requêtes sur le serveur distant peut impliquer un conflit avec les données sur le serveur. La définition d'un conflit peut changer d'un utilisateur à un autre. Il peut même changer au cours du temps pour le même utilisateur. En ce qui concerne la réconciliation, l'utilisateur peut définir ses préférences pour certains conflits prévus. Une stratégie par défaut est appliquée dans le cas d'un conflit non prévu. Cette stratégie consiste à faire comprendre à l'utilisateur le conflit existant pour qu'il puisse intervenir et éditer les données divergentes.

Je vous représente dans ce qui suit le principe de fonctionnement de Java RMI, la base de mon expérimentation. Bien que j'aie travaillé sur Java RMI, les résultats obtenus peuvent être généralisés et appliqués sur tous les modèles utilisant les appels de procédure à distance.

B. JavaRMI

Le langage Java possède un RPC orienté objet intégré. Il permet à des objets situés dans des espaces d'adressage différents sur des machines distinctes d'interagir d'une manière transparente l'un avec l'autre. Un objet distribué est un objet java 'comme les autres', d'où la simplicité de le manipuler comme tout autre objet java.

Il possède un proxy, représentant de l'objet côté client ; et un skeleton, représentant de l'objet côté serveur.

1. Architecture

A la création de l'objet, un stub et un skeleton (avec un port de communication) sont créés côté serveur.

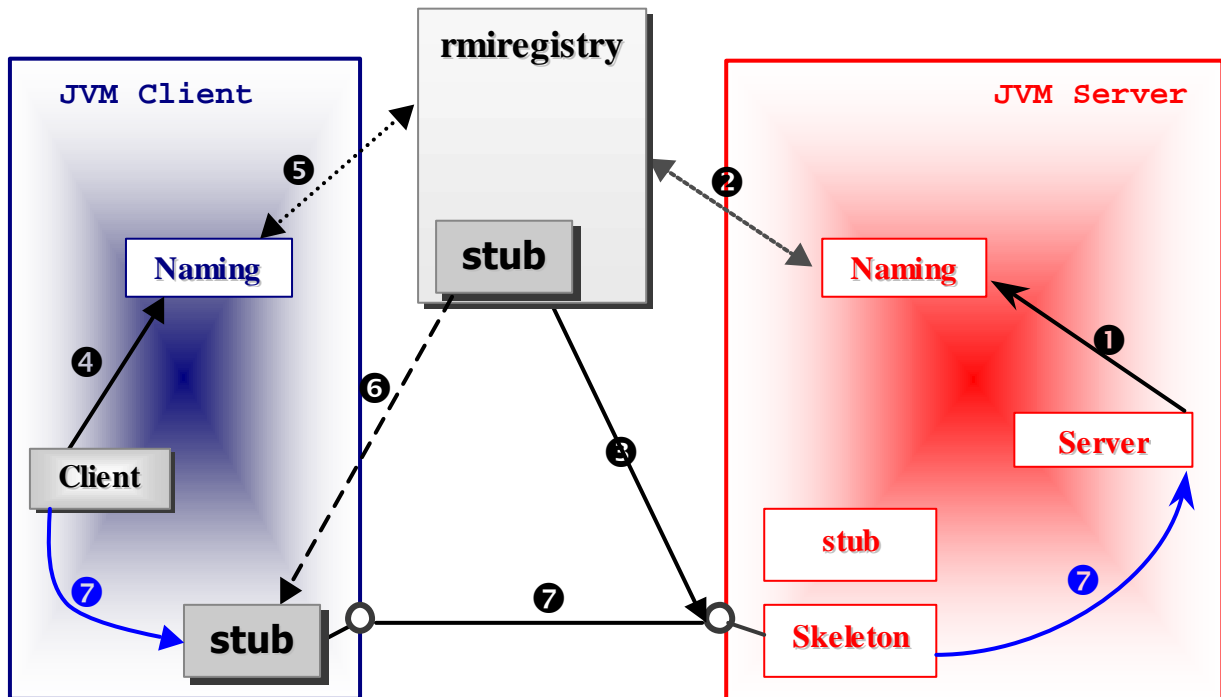


Figure 12 - Mode opératoire du modèle Java RMI

2. Mode Opérateur

- ① L'objet serveur s'enregistre auprès du Naming de sa machine virtuelle, JVM (Java Virtual Machine) en utilisant la méthode rebind.
- ② Le Naming enregistre le stub de l'objet (sérialisé) auprès du serveur de noms (rmiregistry)
- ③ Le serveur de noms est prêt à donner des références à l'objet serveur
- ④ L'objet client fait appel au Naming pour localiser l'objet serveur,..
- ⑤ Le Naming récupère le stub vers l'objet serveur,..
- ⑥ installe l'objet stub et retourne sa référence au client
- ⑦ Le client effectue l'appel à l'objet serveur par appel à l'objet stub

3. Modèle Java RMI adapté aux réseaux sans fils

Le modèle Java RMI exige que la connexion reste maintenue entre le client et le serveur tout au long de la session applicative. Afin d'adapter ce modèle aux réseaux sans fils, on propose d'introduire une entité d'adaptation dans la partie cliente. Cette entité sera responsable de la gestion des envois des requêtes suivant les caractéristiques de l'environnement de la connexion (disponibilité du serveur, disponibilité des ressources, etc..).

Le modèle que nous proposons consiste à adapter le modèle Java RMI aux problèmes de déconnexions fréquentes. On introduit alors une entité adaptative dans la partie cliente que l'on nomme *ConnectionManager*. Le client dans le modèle RMI ne ferait plus ses appels de méthodes directement sur le stub mais sur l'objet *ConnectionManager*. Cet objet s'occupe d'invoquer l'exécution des méthodes sur le stub si la connexion avec le serveur distant est disponible ou à une copie locale du serveur sinon.

Etant donné que la déconnexion du serveur pourra être brusque ou prévue par l'utilisateur de l'application. On peut distinguer deux scénarios :

1. Scénario dans le cas d'une déconnexion prévue

Dans le cas d'une déconnexion prévue par l'utilisateur, l'objet *ConnectionManager* crée dynamiquement une copie (locale ou partielle) du serveur. A chaque fois que le client appelle une méthode, l'objet *ConnectionManager* délègue l'exécution de cet appel à la copie locale du serveur.

2. Scénario dans le cas d'une déconnexion brusque

Dans le cas d'une déconnexion brusque non prévue par l'utilisateur, l'objet peut créer un fichier de log. Tous les appels de méthodes faits en mode non connecté seront enregistrés dans ce fichier. Cette approche peut entraîner la modification du client qui normalement attend une réponse après chaque requête. Une autre politique peut consister à créer une copie locale du serveur sur le site client dès le début de la session applicative puis de mettre à jour régulièrement cette copie afin de prendre en compte l'évolution du système distant.

4. Agenda Mobile, Application

L'agenda mobile est un agenda traditionnel simple adapté pour les utilisateurs qui utilisent des terminaux mobiles et des réseaux sans fils.

Elle consiste en un serveur qui permet de gérer les rendez-vous de ses clients. On peut imaginer un directeur et une secrétaire qui accèdent au même agenda (celui du directeur).

La secrétaire lit la liste des rendez-vous déjà réservés, ajoute ou supprime éventuellement un rendez-vous.

Le directeur peut avoir une copie sur son PDA, son ordinateur portable, sa machine fixe : Il peut mettre à jour les copies de l'agenda sur les différents dispositifs. Il prévoit de travailler en mode non connecté alors il peut sélectionner une partie des rendez-vous à télécharger sur sa machine. La sélection se fait suivant des critères concernant les dates (Les rendez-vous entre deux dates, les rendez-vous supérieurs à une date donnée ou les rendez-vous inférieurs à une date donnée). La sélection pourra se faire également suivant des critères concernant les personnes avec lesquels des rendez-vous sont déjà pris ou suivant les locations des rendez-vous.

1. Implémentation en Java RMI

Le challenge de cette implémentation est de bien définir les stratégies du fonctionnement en mode non connecté ainsi que les politiques de réconciliation dans le cas d'un conflit constaté lors de la diffusion des différentes copies de l'agenda.

On distingue dans cette application deux aspects : l'aspect fonctionnel et l'aspect non fonctionnel.

L'aspect fonctionnel concerne le fonctionnement interne de l'agenda : la gestion des rendez-vous, l'interface graphique, etc..

L'aspect non fonctionnel concerne des aspects qu'on suppose commune à toutes les applications tels la gestion de la consistance entre les différentes copies de l'agenda, le fonctionnement en mode non connecté, la détection des conflits et leur résolutions, etc..

1.1. Définition des interfaces

Afin de distinguer entre les deux aspects fonctionnels et non fonctionnels de notre application, on définit deux interfaces :

- L'interface fonctionnelle qui contient toutes les signatures de méthode nécessaire pour le fonctionnement de l'application.
- L'interface non fonctionnelle qui contient toutes les méthodes nécessaires pour assurer la persistance de l'application.

On définit deux interfaces complémentaires dans l'aspect non fonctionnel : Une interface sur le serveur qui doit permettre de récupérer une copie de serveur et déterminer le protocole de

réconciliation entre les différentes copies de l'agenda ; et une autre interface sur la copie locale qui doit permettre d'enregistrer toutes les actions soumises en mode non connecté.

Les figures ci-dessous représentent les diagrammes de classes côté client et côté serveur

1.2. Diagramme de classes - Côté client

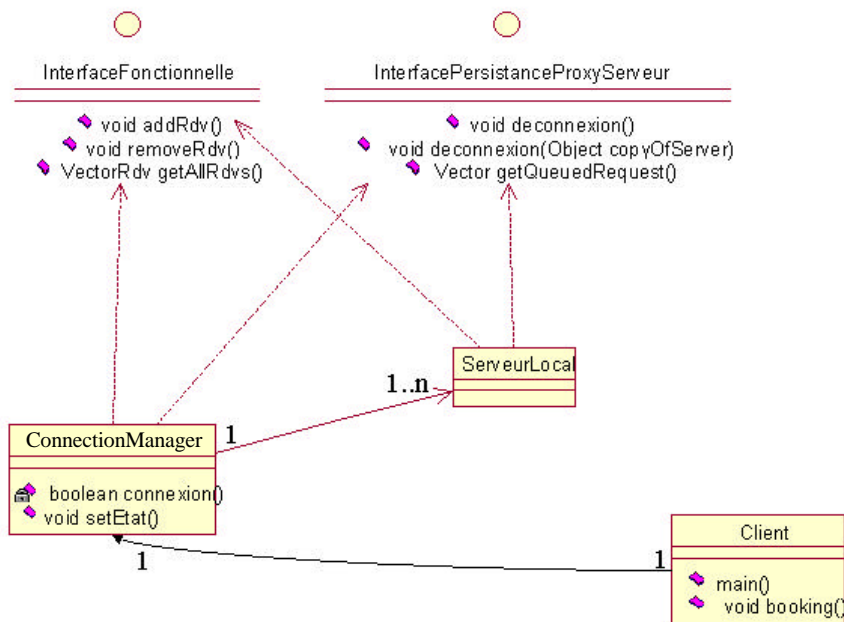


Figure 13 - Diagramme de classes côté client

1.3. Diagramme de classes - Côté serveur

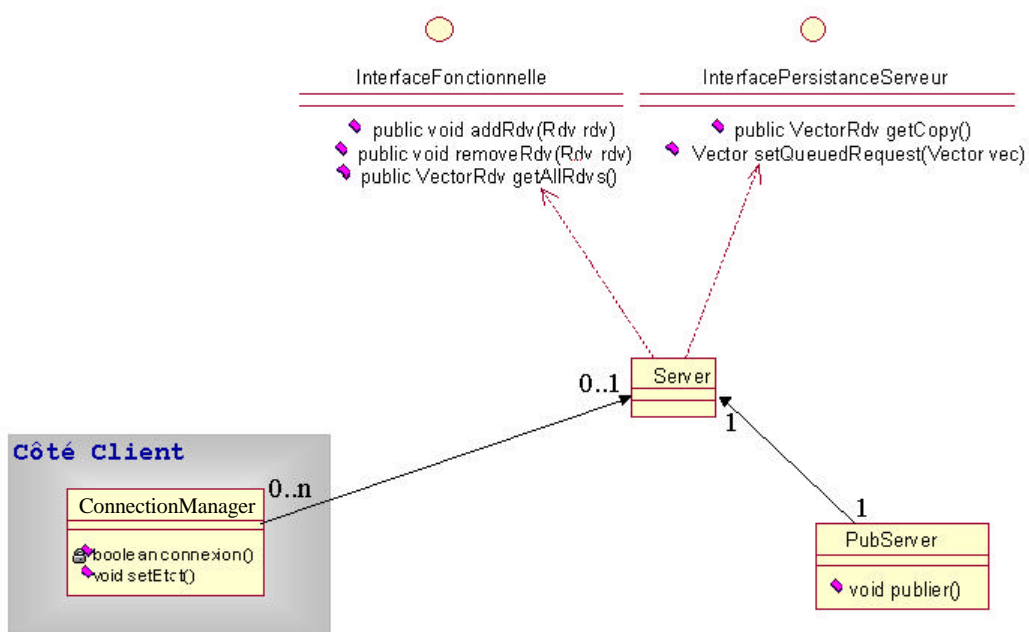


Figure 14 - Diagramme de classes côté serveur

1.4. Fonctionnement en mode déconnecté

Dans le cas d'une déconnexion prévue par l'utilisateur, Une copie du serveur sera récupérée sur la poste client. Le client se connecte sur cette copie locale. Toutes les requêtes remises au serveur distant seront servis localement par cette copie et seront enregistrées dans un fichier de log.

La figure ci-dessous montre les signatures de méthodes appelées côté serveur et côté client.



Figure 15 - Appel de méthodes sur le site client et le site serveur en mode déconnecté

1.5. Phase de réconciliation

Lors de la re-connexion au serveur, toutes les actions déjà enregistrées dans le fichier de logs sur la partie cliente doivent être exécutées sur le serveur. Dans le cas d'un conflit, on fait appel à la politique de la réconciliation. On envoie à l'utilisateur les conflits constatés et qui n'ont pas été résolus. Par exemple, Le directeur peut avoir une priorité plus grande que celle de son secrétaire c'est à dire si le directeur essaye de réserver un rendez-vous sur une plage horaire déjà réservée par son secrétaire. C'est le rendez-vous réservé par la secrétaire qui sera annulée et elle en sera informée. La copie locale du serveur sera supprimée.

La figure ci-dessous montre les signatures de méthodes appelées côté serveur et côté client

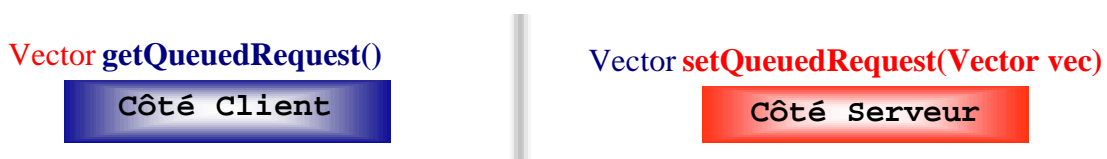


Figure 16 - Appel de méthodes sur le site client et le site serveur en phase de réconciliation

1.6. Exigences des utilisateurs

On note que chaque utilisateur de notre application peut avoir ses propres choix et ses propres préférences en ce qui concerne :

1. Le fonctionnement en mode non connecté :
 - Téléchargement d'une copie totale ou d'une copie partielle ?
 - Choix des données à sélectionner dans le cas d'une copie partielle ?
2. La définition des conflits :

- Le chevauchement de certains rendez-vous ne représente pas un conflit pour tous les utilisateurs.
3. La politique de réconciliation à suivre :
- Des préférences prédéfinies peuvent être envisagées. Mais le même utilisateur peut changer ses choix au cours du temps.

On constate alors que le vrai challenge est de permettre aux utilisateurs de choisir dynamiquement les paramètres contrôlant les différents aspects de leur application. Notre solution est de choisir l'outil NOAH qui permet de connecter **dynamiquement** des objets distants hétérogènes. Le code exprimant cette connexion, que l'on appelle interaction, est complètement **indépendante** et **externe** du code des objets.

2. Vers une implémentation dynamique et adaptée aux réseaux sans fils

2.1. Outil Noah

L'outil Noah est un outil développé au sein de l'équipe Rainbow au laboratoire I3S. Il est basé sur un langage spécifique à la description des interactions nommé, ISL (Interaction Specific Language). Ce langage repose sur un modèle d'interactions qui permet de modifier le comportement des objets à la réception de messages. Ce modèle offre une indépendance entre le code des objets et celui décrivant leurs interactions ce qui facilite la maintenance et l'évolution dynamique séparées de ces deux types de codes et leur réutilisation. Il permet de décrire la connexion d'objets hétérogènes par des interactions n-aires, non orientées. Il est basé sur la définition d'opérateurs (comme séquence, concurrence, etc..), dont la sémantique a été définie, et il s'inspire du langage d'interface IDL (Interaction Définition Language) [LAM, 1987] défini par la norme CORBA.

On introduit deux notions de base dans cet outil : les schémas d'interaction et les interactions. Au niveau des classes, les schémas d'interactions décrivent des connexions possibles entre classes. Au niveau des instances, les interactionsinstancient ces schémas pour représenter les connexions réelles entre les objets.

Le langage ISL offre une syntaxe assez simple pour décrire un schéma d'interaction sous forme de règles.

Une règle ISL décrit comment le comportement d'un objet (c'est à dire la réception d'un message par cet objet) doit être modifié quand il interagit avec les autres. Dans un langage à objets, les comportements correspondent aux méthodes.

Une règle modifie donc le comportement attendu d'une méthode : Une règle est composée d'une partie gauche qui décrit une réception d'un message et d'une partie droite, qui décrit le nouveau comportement à exécuter en utilisant le langage ISL.

Grâce aux schémas d'interaction, ISL offre un moyen d'exprimer à un haut niveau d'abstraction la sémantique de la communication. En particulier, la notion de règle d'interaction permet de modéliser les modifications de comportement des objets interagissants. En plus, les schémas d'interaction peuvent être vus comme des patrons de conception [GAM, 1995] pour les interactions entre objets. Ils modélisent la façon de prendre en compte une interaction du langage d'interface IDL. Cette modélisation est ensuite appliquée différemment selon l'environnement de programmation dans lequel on le projette.

La mise en œuvre, l'ajout ou la suppression d'une interaction pourra être faite d'une façon **dynamique** entre les instances.

2.2. Agenda Mobile et l'outil Noah

L'introduction de l'outil Noah dans l'application Agenda Mobile permet aux utilisateurs de :

- Créer dynamiquement des proxys sur le site client
- Sélectionner dynamiquement le contenu des proxys
- Choisir dynamiquement les stratégies de fonctionnement en mode déconnecté
- Choisir dynamiquement les politiques de réconciliation
- Proposer des scénarios personnalisés adaptés à leurs utilisations

2.3. Exemple d'utilisation des interactions dans Agenda Mobile

On présente ci-dessous une interaction entre le stub et le serveur. Cette interaction permet à l'utilisateur de travailler en mode non connecté. Un proxy représentant une copie locale du serveur distant sera créé dynamiquement.

```
interaction deconnexion(Object stub, Object server) implements Proxy{  
    this.init() -> this.load(server.getCopy());  
    stub.*    -> delegate this._call  
}
```

On note que cette interaction pourra être mise en œuvre d'une façon dynamique entre deux classes déjà existantes. En d'autres mots, l'utilisation de cette interaction permet de gérer l'aspect de déconnexion de toute application déjà écrite en Java RMI sans réécrire le client ou le serveur.

V. Conclusion

Une application répartie est aujourd'hui construite par assemblage de composants en essayant de favoriser au mieux la réutilisation de composants existants. Elle relève d'aspects fonctionnels (la mission que l'utilisateur souhaite voir remplir par l'application), et non fonctionnels (une qualité de service garantie malgré des conditions d'exploitation changeantes). Cette recherche trouve ses origines dans l'incapacité des systèmes actuels à satisfaire les besoins très volatiles des utilisateurs. Cette incapacité se manifeste tant en terme de coûts des adaptations (liés à une productivité insuffisante) que de réactivité (les délais sont généralement tels que les besoins ont évolué entre leur identification et la mise en place d'une solution).

C'est dans le cadre de cette double approche de la problématique de la réutilisation et de l'adaptabilité des systèmes logiciels que ce travail pourra être poursuivi :

Une des perspectives de ce travail consiste à définir une bibliothèque d'interaction adaptée au modèle Java RMI mobile. Cette bibliothèque comprendra des schémas d'interaction abstraits communs à toutes les applications ; et une (ou des) bibliothèque(s) dédiée(s) qui comprennent des schémas d'interaction spécifiques à une application donnée.

Une autre perspective consiste à proposer une nouvelle version du modèle Java RMI qui intégrera l'outil NOAH afin de proposer un nouveau modèle dynamique et adaptée aux réseaux sans fils. Les limitations d'ordre matériel et logiciel imposées par la mobilité doivent être gérés d'une façon transparente à l'utilisateur. Ces apports ne doivent pas nécessiter la réécriture des applications déjà existantes.

Une perspective à long terme consiste à apporter de la conscience à la mobilité à la partie serveuse de l'application (Application aware, System aware) afin d'optimiser la possibilité de l'adaptation à l'environnement et pour autoriser le partage des fonctionnalités entre le client et le serveur.

Finalement, on note que les résultats obtenus de cette expérimentation en Java RMI peuvent être appliqués à tout modèle client – serveur basé sur un appel de procédure à distance.

VI. Bibliographie

[André et Segarra, 1999]

F. André, M.T. Segarra. *A Generic Approach to Build Mobile Applications.* INRIA Research Report No. 3723, June 1999.

[André et Segarra, 2000]

F. ANDRE et M.T. SEGARRA. *A Generic approach to satisfy Adaptability needs in Mobile Environments.* Proc. of the 33rd Annual Hawaii International Conference on System Sciences (HICSS33), January 2000, Maui, Hawaii, USA.

[Demers et al., 1994]

A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. *The Bayou Architecture: Support for Data Sharing among Mobile Users.* In Workshop on Mobile Computing Systems and Applications, Sta. Cruz, CA, USA, Dec. 1994

[Dery et al., 2002]

A.-M. Dery, M. Blay-Fornarino et S. Moisan. *Apport des interactions pour la distribution des connaissances.* Objet, Systèmes distribués et connaissances. 2002

[Fox et Brewer, 1996]

A. Fox et E. Brewer. *Reducing WWW Latency and bandwidth requirements by Real-Time Distillation.* In Proc. Of the 5th International WWW conference, Paris, France, May 1996.
http://www5conf.inria.fr/fich_html/papers/P48/Overview.html

[Houssel et al., 1998]

B. Housel, G. Samaras, and D. Lindquist. *WebExpress: A Client/Intercept Based System for Optimizing Web Browsing in a Wireless Environment.* Mobile Networks and Applications, 3:419–431, 1998.

[Jing et al., 1999]

J. Jing, A. Helal, A. ElMagarmid. *Client-Server Computing in Mobile Environments.* ACM Computing Surveys, Vol. 31, N°2, June 1999

[Kermarrec et al., 2001]

A.-M. Kermarrec, A. Rowstron, M. Shapiro et P. Druschel, *The IceCube approach to the reconciliation of diverging replicas*. A paraître dans proceedings of the 20th annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC) Aug. 2001.

[Kistler et Satyanarayanan, 1992]

J. Kistler, M. Satyanarayanan. *Disconnected Operation in the Coda File System*. *ACM Transactions on Computer Systems*, 10(1) :3-25, February 1992.

[Le et al., 1994]

M. Le, S. Seshan, F. Burghardt, et J. Rabaey, *Software architecture of the InfoPad system*. In Proceedings of the Mobidata Workshop on Mobile and Wireless Information Systems. Nov. 1994.

[Liu et al., 1995]

G. Liu, A. Marlevi, and G. M. JR. *A Mobile Virtual distributed System Architecture for Supporting Wireless Mobile Computing and Communications*. In Proc. of the 1st Annual International Conference on Mobile Computing and Networking, 1995.

[Marangozova et Hagimont, 2002]

V. Marangozova et D. Hagimont. *Non functional replication management in the Corba Component Model*. In Proceedings of the 8th International IFIP/ACM Conference on Object-Oriented Information Systems, Montpellier (France), September, 2002.

[Noble et al., 1997]

B. Noble, M. Satyanarayanan, D. Narayanan, J. Titlon, J. Flinn and J. Walker. *Agile Application-Aware Adaptation for Mobility*. In Proc. Of the 16th symposium on operating systems principles, st. Malo, France, October 1997.

[Shapiro et al., 2000]

M. Shapiro, A. Rowstron et A.-M. Kermarrec. *Application-independent reconciliation for nomadic applications*. SIGOPS European Workshop on "Beyond the PC: New Challenges for the Operating System", Kolding (Denmark), Sept. 2000.

[Terry et al., 1998]

D. Terry, K. Petersen, M. Spreitzer, and M. Theimer. *The case for non-transparent Replication : Examples from Bayou.* IEEE Data Engineering, December 1998, pages 12-20.

[Terry et al., 1994]

D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, et B. Welch. *Session guarantees for weakly consistent replicated data.* In Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems. PDIS, Austin, TX, Sept. 1994.

[Terry et al., 1994]

D.B. Terry, M.M. Theimer, K. Petersen, A.J. Demers, M.J. Spreitzer, et C.H. Hauser. *Managing update conflicts in Bayou, a weakly connected replicated storage system.* ACM SIGOPS Oper. Syst. Rev. 29, 5, 172–182, Dec. 1995.

[Watson, 1994]

T. Watson. *Wit : An infrastructure for wireless palmtop Computing.* Technical report UW-CSE-94-11-08, Departement of Computer Science and Engineering. University of Washington, 1994. Available at : <http://www.cs.washington.edu/research/tr/tr-by-date.html>

[Zenel et Duchamp, 1997]

B. Zenel et D. Duchamp. *A General Purpose Proxy Filtering Mechanism applied to the Mobile Environment.* In proc. Of the 3rd Annual International Conference on Mobile Computing and Networking, Budapest, Hungary, Septembre 1997.