État de l'art de la gestion de cache logiciel pour tolérer les déconnexions en environnements mobiles
Survey of software cache management for tolerating disconnections in mobile environments

Nabil Kouici, Denis Conan and Guy Bernard
GET / INT, CNRS Samovar
9 rue Charles Fourier
91011 Évry, France
{Nabil.Kouici, Denis.Conan, Guy.Bernard}@int-evry.fr

October 17, 2006

Résumé

Ces dernières années ont été marquées par une forte évolution des équipements et des réseaux utilisés dans les environnements mobiles. Cette évolution a abouti à la définition d'une nouvelle thématique: l'informatique mobile. L'informatique mobile offre aux utilisateurs la capacité de pouvoir se déplacer tout en continuant à accéder aux applications réparties, ceci indépendamment de la localisation géographique. Cependant, l'informatique mobile soulève le problème de la disponibilité de ces applications en présence de déconnexions. Ainsi, les applications réparties et les intergiciels fonctionnant dans les environnements mobiles doivent s'adapter aux déconnexions. Dans cet article, nous présentons un état de l'art de la gestion de cache d'entités logicielles pour la tolérance aux déconnexions dans les environnements mobiles. L'article est structuré selon le type d'entités mises en cache: fichiers ordinaires, bases de données, pages web, objets, et enfin composants.

Abstract

Recent years have seen a rapid evolution in the hardware and in networks used in mobile computing environments. This evolution has opened up new opportunities for mobile computing. Mobile computing allows a mobile user to access various kinds of information anytime anywhere. However, mobile computing raises the problem of data availability in

the presence of disconnections. Thus, distributed applications and middleware in mobile environments must be adapted to cope with disconnections. In this article, we present a state of the art on software caching for tolerating disconnections in mobile environments. The structure of the article follows the type of entities manipulated by the distributed applications : ordinary files, databases, web pages, objects, and components.

I Introduction

Les récentes avancées dans le domaine des communications sans fil aussi bien du point de vue des réseaux (GSM, GPRS, UMTS, Wi-Fi, etc.) que de celui des terminaux (PC portables, assistants personnels numériques, téléphones portables, etc.) rendent possibles de nouvelles applications dans lesquelles l'utilisateur peut avoir accès à l'information à n'importe quel moment et depuis n'importe quel endroit. Ainsi, un nouveau paradigme est apparu connu sous le nom d'informatique mobile. L'informatique mobile se caractérise par le nomadisme des utilisateurs qui introduit une forte variabilité des ressources disponibles: bande passante, batterie, processeur, mémoire, etc. Cette variabilité ne doit pas être vue comme une faute puisqu'elle est une conséquence de la mobilité des utilisateurs, même si elle mène à des déconnexions. Nous considérons deux types de déconnexions: les déconnexions volontaires et les déconnexions involontaires. Les premières, décidées par l'utilisateur depuis son terminal mobile, sont justifiées par les bénéfices attendus sur le coût financier des communications, l'énergie, la disponibilité du service applicatif, et la minimisation des désagréments induits par des déconnexions inopinées. Les secondes sont le résultat de coupures intempestives des connexions physiques du réseau, par exemple, lors du passage de l'utilisateur dans une zone d'ombre radio.

Dans une application répartie « classique » fonctionnant avec une bonne connectivité entre les différentes entités[1], les clients peuvent s'exécuter sur des terminaux légers et n'être constitués que d'une interface graphique, les entités rendant les services restants sur des nœuds du réseau fixe. Pour assurer la continuité de service lors de déconnexions, des mandataires des entités « serveurs » doivent être instanciés sur les hôtes des clients, les opérations pendant les phases de déconnexion doivent être journalisées et des réconciliations doivent être effectuées lors des reconnexions. Ainsi, tout intergiciel ou système d'exploitation supportant des applications en environnements mobiles doit proposer les quatre services système suivants : détection des déconnexions [64], mise en cache d'entités logicielles pendant les périodes de bonne connectivité réseau [18], journalisation d'états ou d'opérations pendant les déconnexions, et réconciliation à la reconnexion pour assurer la cohérence globale des applications réparties [57]. Le premier mécanisme, la détection de déconnexion, est indépendant du type d'application. Par contre, les trois autres mé-

[1]Dans cet article, le terme « entité » est le terme générique pour désigner un fichier, une base de données, une page web, un objet ou un composant. En outre, les différents niveaux de connectivité sont définis plus loin dans l'article; ici, « bonne connectivité » s'entend dans son sens commun.

canismes sont inter-dépendants: par exemple, la granularité des entités mises en cache contraint le type de journalisation et la méthode de réconciliation.

Les états de l'art existants dans la littérature se focalisent sur certaines granularités comme les systèmes de fichiers [18], les objets répartis [40], ou encore les bases de données [17]. Dans cet article, nous dressons un panorama englobant toutes les orientations (paradigmes) utilisées pour la conception d'applications réparties: les fichiers ordinaires, les bases de données, les pages web, les objets et les composants. En outre, les travaux précédents ne prennent en compte qu'un sous-ensemble de critères et ne détaillent pas spécifiquement la gestion du cache [27]. Au contraire, nous étudions les solutions de la littérature dédiées à la gestion de cache selon neuf critères que nous présentons: la granularité, le type de déconnexion (volontaire ou involontaire), les modes de fonctionnement possibles (connecté, partiellement connecté ou déconnecté), la stratégie d'adaptation (laisser-faire, transparence ou collaboration), le type d'adaptation (statique, dynamique ou auto-adaptation), les stratégies de déploiement et de remplacement, la proposition ou non d'un modèle de conception, et la prise en compte des dépendances entre les entités de l'application. En outre, bien que certaines solutions soient plus prometteuses pour les recherches à venir, nous ne pensons pas que certaines orientations (par exemple, l'orientation fichier) ne trouvent pas de domaines applicatifs dans lesquelles elles sont efficaces et efficientes. Par exemple, les systèmes de fichiers sont bien adaptés pour des applications de travail collaboratif à base de documents de texte ou d'images. Par conséquent, toutes les orientations peuvent trouver leur pertinence dans un domaine applicatif donné. Enfin, nous écartons de l'étude les applications imposant des contraintes de temps, que ce soit du temps réel mou ou dur. Nous ne considérons donc pas les applications dites temps-réel, ni les applications multimédias à flot continu. La raison est que les mécanismes mis en œuvre pour ces applications sont spécifiques: en plus de la collaboration entre l'application et le système, une collaboration est nécessaire entre le système et le réseau. Cette collaboration demande par exemple des protocoles de réservation de ressources, et de contrôle et de maintien de la qualité de service.

L'article est organisé comme suit. Nous débutons l'étude dans la section II par la description des besoins en termes d'adaptation aux déconnexions et les différents modes de fonctionnement des terminaux mobiles. Ensuite, nous introduisons le concept d'opération déconnectée dans la section III, puis énumérons nos critères d'étude dans la section IV. Dans la suite, l'état de l'art est organisé selon le critère le plus discriminant : la granularité. Nous présentons donc successivement les solutions de la littérature pour les orientations fichiers, bases de données, pages web, objets et composants, respectivement dans les sections V, VI, VII, VIII et IX. Enfin, nous synthétisons les différentes approches dans la section X.

II Besoins d'adaptation et modes de fonctionnement

Nous présentons dans cette section les besoins d'adaptation des applications réparties pour tolérer les déconnexions (section II.1), puis, nous détaillons les différents modes de fonctionnement des applications dans les environnements mobiles (section II.2).

II.1 Besoins d'adaptation

Selon le dictionnaire Le Petit Robert, l'adaptation est « l'aptitude [d'un système] à modifier sa structure ou son comportement pour répondre [...] à des situations nouvelles ». En se basant sur ce principe, l'adaptation aux déconnexions est la capacité d'un système à s'ajuster pour continuer à fonctionner en présence de déconnexions. Une adaptation se définit par une stratégie, un type, des politiques et des mécanismes. La stratégie d'adaptation répond à la question « qui intervient dans l'adaptation? » et fait le lien entre le type, les politiques et les mécanismes. Le type d'adaptation précise à quel moment l'adaptation doit être effectuée, les politiques expriment les mécanismes à mettre en œuvre et les mécanismes opérent les actions d'adaptation.

D'après [58], l'intervalle des stratégies d'adaptation est délimité par deux extrémités. Dans la stratégie *laisser-faire*, l'adaptation est entièrement de la responsabilité des applications. Cela évite de recourir à un support système (système d'exploitation ou intergiciel). À l'autre extrémité, la stratégie *transparence* fait en sorte que l'adaptation soit entièrement de la responsabilité d'un intergiciel ou d'un système d'exploitation. Entre ces deux extrêmes, il existe une autre stratégie d'adaptation appelée *collaboration* dans laquelle l'application et l'intergiciel se concertent. L'intergiciel surveille les ressources, signale aux applications tout changement significatif, et fournit les mécanismes d'adaptation. L'application, quant à elle, fournit les politiques d'adaptation. Les nombreux travaux synthétisés dans [27] montrent que les stratégies laisser-faire et transparence ne sont pas adéquates pour supporter les déconnexions. En effet, dans la première stratégie, il manque un contrôleur central arbitrant les demandes de ressources concurrentes et incompatibles[2], et également les aspects extrafonctionnels que doivent gérer les développeurs. Dans la deuxième stratégie, certaines adaptations peuvent être insatisfaisantes ou même contre-productives pour certaines applications car elles ne prennent pas en compte la sémantique des applications. Dans cette étude, nous introduisons l'utilisateur comme troisième acteur de l'adaptation, en plus de l'application et de l'intergiciel. L'utilisateur peut intervenir de trois façons différentes [60]: en guidant l'application pour changer son comportement, en demandant à l'intergiciel de garantir un certain niveau de ressources, et enfin en répondant à des suggestions de l'intergiciel ou de l'application (par exemple, l'intergiciel peut demander à l'utilisateur d'autoriser la transition d'un réseau Wi-Fi non payant vers un réseau GPRS payant).

[2]Chaque application ne connaît que les demandes qu'elle effectue et considère qu'elle est la seule cliente.

Concernant le type d'adaptation, [6] propose une classification en trois catégories. L'adaptation statique intervient pendant le développement ou avant l'exécution, soit en modifiant le code, soit en modifiant les options de compilation ou de déploiement; elle nécessite de connaître *a priori* l'environnement d'exécution de l'application. L'adaptation dynamique intervient pendant l'exécution de l'application; elle est réalisée par une intervention extérieure, la plupart du temps humaine. Enfin, l'auto-adaptation intervient également pendant l'exécution, mais peut être initiée par l'application ou par l'intergiciel. Les choix sont par exemple effectués de manière automatique par l'intergiciel à partir de politiques fixées par l'application.

II.2 Modes de fonctionnement

La figure 1 schématise les différents modes de fonctionnement des terminaux mobiles (connecté, partiellement connecté, déconnecté et veille) ainsi que les différentes transitions possibles entre ces modes [51]. Dans le mode connecté, le terminal mobile dispose d'une bonne connexion au réseau, à la manière d'une station fixe. Dans le mode partiellement connecté, le terminal mobile ne dispose plus pour communiquer avec le réseau que d'un lien faible à la suite, par exemple, de perturbations du réseau hertzien ou d'un faible niveau de la batterie. Dans le mode déconnecté, le terminal mobile est isolé soit parce qu'il n'est plus physiquement relié au réseau soit parce qu'il est impossible de maintenir une connexion sans fil. Enfin, dans le mode veille utilisé par les terminaux mobiles pour préserver les ressources énergétiques, la connexion réseau est maintenue, mais le terminal mobile ne s'en sert pas. L'objectif de notre étude étant la gestion des déconnexions, nous ignorons le mode veille et ne considérons que les modes connecté, partiellement connecté et déconnecté.

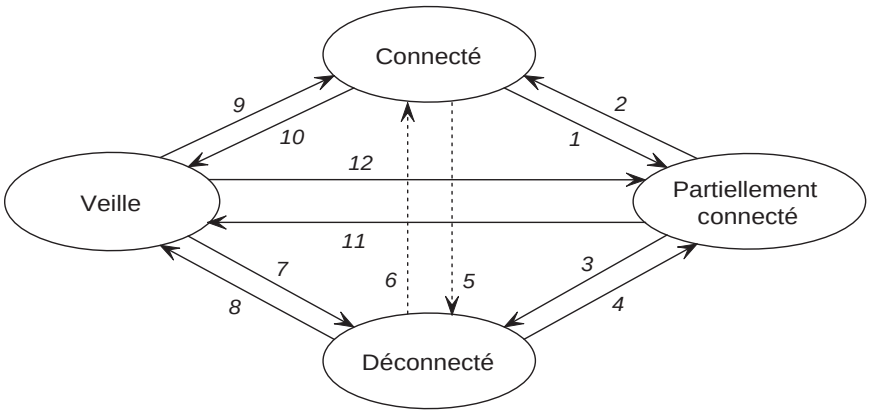

Figure 1: Modes de fonctionnement des applications.
Application's functioning mode.

Dans la figure 1, les transitions entres les modes de fonctionnement correspondent aux changements de valeur de la disponibilité de la ressource considérée. Les pseudo-transitions *5* et *6* correspondent à des déconnexions/reconnexions brusques. Ce sont des pseudo-transitions car le passage entre les modes de fonctionnement connecté et déconnecté ne se fait pas directement. En effet, nous passons toujours par le mode partiellement connecté qui est, dans ce cas, un état transitoire de courte durée.

Dans les environnements mobiles, la détection de la connectivité peut être assurée par un détecteur de connectivité [64]. Ce dernier donne des informations sur une ressource telle que la bande passante ou la batterie pour déterminer localement si le terminal mobile peut ou non utiliser telle ou telle interface réseau. La prédiction du niveau de ressource du terminal mobile joue un rôle clé dans l'établissement du mode de fonctionnement. Dans la problématique de gestion de la carence des ressources de communication, la prédiction permet d'anticiper les besoins de chargement du cache en prévision de déconnexions. Par contre, dans une situation de très bonne connectivité, elle pourrait être utilisée pour décider le lancement d'exécutions distantes (en anglais, *off-loading*) afin de préserver les ressources locales (processeur, batterie, mémoire, etc.) [54, 16, 20].

III Concept d'opération déconnectée

La continuité de service en présence de déconnexions est assurée par le maintien d'une connexion logique en utilisant le concept d'opération déconnectée [32, 44] illustré dans la figure 2. Quatre principaux acteurs interviennent: le client (l'application côté utilisateur) sur l'hôte mobile, l'entité distante de l'application répartie, le cache logiciel qui représente une partie de la mémoire de l'hôte mobile, et enfin, l'entité déconnectée qui est un mandataire en termes de code et d'état de l'entité distante. Le fonctionnement des opérations déconnectées est le suivant. En présence des déconnexions, le client sur le terminal mobile accède à l'entité déconnectée préalablement déployée dans le cache. Les opérations effectuées sur l'entité déconnectée pendant les phases de déconnexions sont journalisées et des réconciliations lors des reconnexions sont opérées entre l'entité distante et l'entité déconnectée. Dans le schéma général, le client peut aussi être un serveur pour d'autres clients et le serveur peut également s'exécuter sur un hôte mobile.

Le gestionnaire du cache doit offrir deux stratégies [18]. La stratégie de déploiement détermine les entités déconnectées à créer dans le cache du terminal mobile, quand et pour quelle durée. La stratégie de remplacement décide quelles entités déconnectées doivent être supprimées lorsqu'il n'existe plus assez d'espace mémoire. Par ailleurs, l'utilisation d'une entité déconnectée peut exiger la présence d'autres entités déconnectées dans le cache. Ainsi, les stratégies de déploiement et de remplacement doivent prendre en compte les dépendances entre les entités de l'application. Notons que dans le cas des terminaux mobiles possédant, en plus d'une mémoire vive, un disque dur, ce supplément d'espace mémoire est géré de manière automatique, via le système d'exploitation, par les stratégies

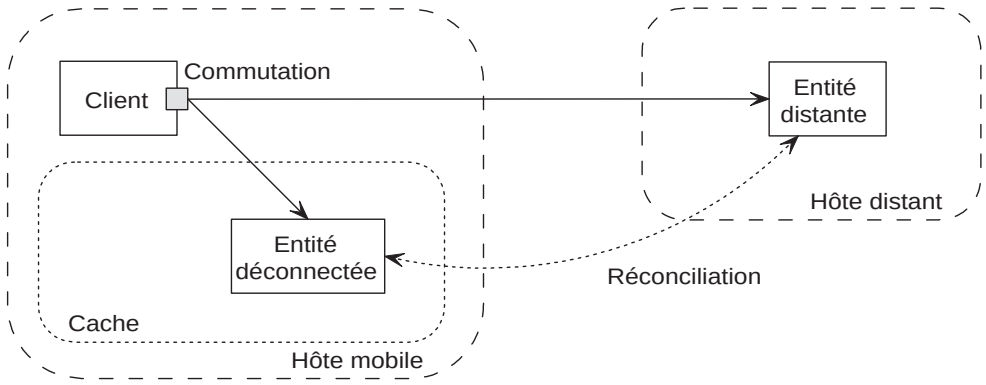

Figure 2: Fonctionnement des opérations déconnectées.
Functioning of disconnected operations.

de déploiement et de remplacement.

IV Critères d'étude des solutions de gestion du cache pour les déconnexions

Avant d'aborder les principales solutions de gestion du cache pour les déconnexions, nous présentons les différents critères de classification que nous avons retenus. Dans cet article, nous proposons une classification selon neuf critères:

- la granularité: définit le type de l'entité manipulée par l'application, soit fichier, base de données, page web, objet ou composant;
- le type de déconnexion: volontaire ou involontaire;
- le mode de fonctionnement: selon la section II.2, connecté, partiellement connecté et déconnecté;
- la stratégie d'adaptation: selon la section II.1, laisser-faire, transparence ou collaboration;
- le type d'adaptation: selon la section II.1, statique, dynamique et auto-adaptation;
- les stratégies de déploiement et de remplacement du cache: définissent les procédures d'élection pour la mise en cache et d'éviction du cache;
- la modélisation des déconnexions lors de la conception: indique si la solution propose un modèle de conception d'application devant fonctionner en présence des déconnexions;

- la prise en compte des dépendances entre entités de l'application: définit si les dépendances entre les entités de l'application sont prises en compte dans les stratégies de déploiement et de remplacement du cache.

Nous n'incluons pas le type de cache: local (au terminal mobile) ou réparti (et donc partagé par un ensemble de terminaux), dans la liste car, dans la suite de cet article, ne sont abordés que les travaux de recherche sur la gestion locale de caches locaux en vue de la tolérance aux déconnexions.

Dans la suite de l'article, nous présentons les solutions suivant le critère le plus discriminant: la granularité, qui représente l'entité de programmation manipulée par l'application. Les trois premiers types de systèmes (orientés fichier, base de données et page web) définissent la granularité des données (persistantes), tandis que les deux suivants (orientés objet et composant) définissent plutôt la granularité d'association des données avec les traitements. Il est donc possible d'avoir un système orienté fichier et composant. Nous donnons pour chaque granularité les solutions les plus significatives de la littérature. La fin de l'article synthétise les différentes solutions présentées (cf. tableau I).

V Systèmes orientés fichier

La gestion des déconnexions a été abordée pour la première fois dans le cadre des systèmes de gestion de fichiers. Ces travaux inroduisent des concepts repris dans les autres orientations. Même si elle est un peu plus ancienne, l'orientation fichier est toujours très présente par exemple dans les logiciels de synchronisation pour petits dispositifs tels que les assistants personnels numériques ou dans des projets de recherche de grande envergure en informatique ubiquitaire tels que Aura [19, 61]. Nous détaillons dans cette section les trois principales propositions de la littérature: Coda, Amigos et Seer. Ensuite, nous résumons les résultats de quelques autres travaux de recherche.

V.1 Coda

Coda [32, 59] hérite des caractéristiques de conception et d'utilisation de AFS (*Andrew File System*) [23]. Les clients visualisent Coda comme un simple système de gestion de fichiers partagés. L'espace de nommage de Coda est partagé sur plusieurs serveurs d'archivage qui construisent des sous-arbres appelés volumes. Puisque Coda utilise la réplication des serveurs pour permettre la disponibilité et la tolérance aux fautes, les serveurs répliqués peuvent aussi être vus comme un cache réparti.

En plus des serveurs de répliques, Coda utilise un cache local sur chaque terminal. Lorsqu'aucun serveur de répliques n'est accessible, Coda utilise les volumes déployés localement. Le contenu du cache local est géré par un gestionnaire de cache appelé Venus. Il peut être dans l'état *accumulation*, *émulation* ou *réintégration*. La figure 3 représente

le diagramme de transitions d'états de Venus. Venus est normalement dans l'état *accumulation*: les fichiers sélectionnés par la stratégie de déploiement sont récupérés à partir des serveurs de répliques. Lors d'une déconnexion, Venus passe dans l'état *émulation* et l'utilisateur continue son travail sur les volumes disponibles dans le cache. La commutation entre les fichiers dans le cache et les fichiers dans les serveurs distants est réalisée par le système de gestion de fichiers; cette commutation est transparente à l'utilisateur. À la reconnexion, Venus passe à l'état *réintégration* et profite du mode partiellement connecté pour opérer une propagation des modifications « goutte à goutte » en tâche de fond pour dégrader le moins possible le temps de réponse des nouvelles demandes [43].

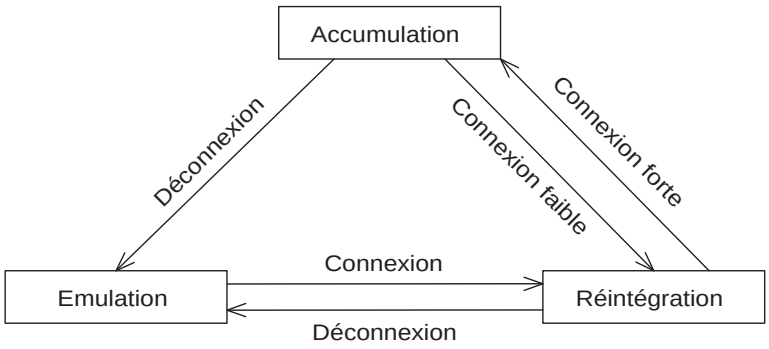

Figure 3: Diagramme de transitions du gestionnaire de cache de Coda.
Transition diagram of the cache manager of Coda.

La stratégie de déploiement de Coda est basée sur l'historique d'utilisation du client. Elles prennent la forme d'une base de données appelée HDB (*Hoard Data Base*) construite par le client de l'application. Un programme appelé *Hoard* permet à l'utilisateur de mettre à jour la HDB directement ou via un script de commandes appelé *Hoard Walking*. En mode accumulation, Coda anticipe les déconnexions en copiant localement (pré-chargement) les fichiers de la HDB selon une priorité accordée par l'utilisateur. Les autres fichiers sont chargés sur un défaut d'accès. Coda utilise la politique LRU (*Least Recently Used*) pour le remplacement.

La dépendance entre fichiers ou volumes n'a pas été abordée dans Coda. En conséquence, si l'utilisateur opère un mauvais choix des fichiers pour la HDB suite à une mauvaise compréhension de l'application, cette dernière peut ne pas fonctionner en mode déconnecté. Enfin, l'absence de données dans le cache ne peut pas être masquée, elle apparaît à l'utilisateur comme une erreur d'accès à un fichier manquant.

V.2 Amigos

Le système de gestion de fichiers Amigos [1] étend NFS (*Network File System*) [49] pour permettre les opérations déconnectées[3]. Les déconnexions peuvent être volontaires et involontaires, mais le mode partiellement connecté n'existe pas. En mode connecté, l'accès aux fichiers reste inchangé par rapport à NFS. En tâche de fond, Amigos déploie des fichiers sur le terminal mobile en se basant sur un profil d'application défini par l'utilisateur. Le profil contient des références de fichiers et de répertoires. L'utilisateur assigne à chaque référence une priorité comme dans Coda. L'utilisateur peut à tout moment changer son profil. En mode déconnecté, toutes les requêtes de l'utilisateur sont redirigées, d'une manière transparente à l'application, vers le cache. Le mécanisme de redirection est intégré dans le système de gestion de fichiers. Dans ce mode, le fichier/répertoire distant est verrouillé pour éviter les mises à jour concurrentes; autrement dit, les autres utilisateurs ne peuvent plus modifier ces entités, voire le verrou est permanent si l'utilisateur est défaillant ou ne se reconnecte pas. Par ailleurs, Amigos ne définit aucune dépendance entre les fichiers/répertoires dans les stratégies de déploiement et de remplacement.

V.3 Ficus/Seer/Roam

Le système Seer [39], basé sur Ficus [21], propose une approche prédictive du déploiement. Elle est basée sur l'idée qu'un système peut observer le comportement de l'utilisateur, faire des inférences sur les relations sémantiques entre les fichiers référencés et utiliser ces inférences pour sélectionner les fichiers à déployer localement. Seer se base sur le concept de distance sémantique pour définir la dépendance entre les fichiers [38]. Les fichiers ayant entre eux une faible distance sémantique sont regroupés en *projets*. Seer définit les distances sémantiques suivantes:

- la distance sémantique temporelle entre deux références à des fichiers est le temps écoulé entre ces deux références;
- la distance sémantique séquentielle entre deux références à des fichiers est le nombre d'accès à d'autres fichiers que les premiers considérés;
- la distance sémantique dite cycle de vie définie entre l'ouverture de A et l'ouverture de B est égale à 0 si A n'est pas fermé avant l'ouverture de B ou égale au nombre d'ouvertures d'autres fichiers (y compris celles de B).

L'architecte de Seer définit deux entités: un observateur et un corrélateur. Le premier observe le comportement de l'utilisateur et ses accès aux fichiers, classant chaque accès selon le type de fichier (texte, vidéo, image...). Ensuite, il fournit ces résultats au

[3]Les auteurs utilisent la formulation « opérations semi-connectées » pour désigner les opérations déconnectées.

corrélateur. Ce dernier calcule les distances sémantiques entre les fichiers et construit les projets. L'entité de déploiement dans Seer étant tout le projet, quand un nouveau contenu de cache doit être choisi, le corrélateur examine l'ensemble des projets pour trouver ceux qui sont actuellement en activité et choisit les projets prioritaires. La procédure de calcul de la priorité entre projets n'est pas fixée dans Seer. En outre, puisque la stratégie de déploiement ne prend en compte que les références aux fichiers déjà visités par l'utilisateur, un fichier non référencé avant la déconnexion n'est pas dans le cache pour le mode déconnecté. Enfin, Seer ne propose pas de stratégie de remplacement.

Le système Seer est utilisé dans le modèle de réplication *Ward* (en français, quartier) mis en œuvre dans Roam [52]. Les déconnexions peuvent être volontaires et involontaires. Les membres d'un quartier peuvent fonctionner en modes déconnecté et connecté, et se déplacer de quartiers en quartiers selon deux schémas: un déménagement (lourd: transfert des données répliquées sur le tuteur du quartier d'arrivée) ou un passage (accès au quartier d'arrivée et au quartier de départ: sans transfert de données, mais avec accès aux données du quartier d'arrivée).

V.4 Autres projets

PFS (*PRAYER File System*) [14] définit une architecture à trois tiers: client sur le terminal mobile, serveur fixe et client fixe. Ce dernier est une réplique du serveur fixe. Contrairement à Coda, le client PFS sur le terminal mobile n'interagit qu'avec le client fixe. En mode connecté, le client PFS utilise directement le client fixe. En modes déconnecté et partiellement connecté, il utilise les fichiers dans le cache local. En outre, en modes connecté et partiellement connecté, PFS propage les modifications effectuées en envoyant les blocs de fichiers modifiés. La propagation des modifications en mode partiellement connecté est aussi utilisée dans [24] qui propose une extension du gestionnaire de cache de PFS pour améliorer la phase de réintégration en classant les trafics (données, vidéos, etc.) et en retardant certaines demandes à réintégrer parce qu'elles deviennent peut-être obsolètes à la vue des opérations qui suivent. Cette dernière idée a aussi été étudiée dans le cadre du projet Odyssey [45].

VI Systèmes orientés base de données

Plutôt que de considérer des fichiers sans structure interne, les bases de données organisent les données en tables, enregistrements et champs. De la même manière, la notion de transaction délimite des portions de code. La gestion du cache et de la réconciliation ne peut en être que plus fine. Les solutions de gestion des déconnexions dans les systèmes orientés base de données ont été étudiées indirectement dans les thématiques de la duplication des bases de données, des transactions mobiles et des bases de données embarquées [4]. Dans cet article, nous nous focalisons sur la duplication des bases de

données dans des terminaux mobiles. C'est pourquoi nous présentons Bayou, la solution de référence dans les environnements mobiles, et ne donnons ensuite qu'une brève présentation d'autres travaux de recherche.

VI.1 Bayou

Bayou [50, 65] cible des bases de données légères qui peuvent être déployées sur des terminaux mobiles avec des systèmes tels que *Sybase Adaptive Server Anywhere*, *Oracle 8i Lite*, *SQLServer* pour Windows CE, *DB2 Everyplace*. Comme représenté dans la figure 4, chaque base de données est répliquée dans plusieurs terminaux, l'un d'eux étant promu serveur primaire. Cette réplication est dynamique à la demande de l'utilisateur. L'application cliente communique avec le serveur à travers une interface spécifique. Cette interface est implantée dans la souche du client et prend en charge la commutation transparente entre la copie locale et les copies distantes de la base de données.

Bayou réalise un schéma de réplication *read-any/write-any*, c'est-à-dire un client peut écrire dans une base de données d'un serveur et lire les données d'un autre serveur. Le client observe donc des divergences, à moins que les deux serveurs aient mis à jour leurs bases de données de façon cohérente. Pour ce faire, Bayou utilise un protocole dit anti-entropique pour la propagation des mises à jour. Les dépendances entre les données apparaissent comme des dépendances entre opérations, elles-mêmes capturées à l'aide d'horloges logiques vectorielles. Le protocole anti-entropique assure que toutes les copies d'une base de données convergent vers le même état, même si les réconciliations s'opèrent deux à deux de manière asynchrone. Le serveur primaire estampille les opérations pour construire un ordre global unique afin de valider (« commit ») les opérations. Une opération non estampillée par le serveur primaire est dite « tentative ». La diffusion des opérations est épidémique et à chaque fois qu'une nouvelle opération est reçue par le serveur primaire, il vérifie les dépendances avec les opérations déjà validées. Pour résoudre un conflit, il peut être nécessaire de défaire une opération « tentative ». En mode connecté (au moins un lien point-à-point est en mode connecté), le client fonctionne en *read-any/write-any*, contrairement à Coda qui n'utilise qu'un seul serveur répliqué. Les serveurs avec lesquels l'application cliente peut communiquer sont choisis suivant un patron d'usage qui dépend du réseau: par exemple, le client n'accède qu'aux serveurs de la même cellule radio. Périodiquement, le protocole anti-entropique propage les écritures. En modes partiellement connecté et déconnecté, le client n'utilise que les bases de données locales, bloquant le client s'il ne dispose pas d'une réplique locale des bases de données. Enfin, le protocole anti-entropique continue à fonctionner en mode partiellement déconnecté.

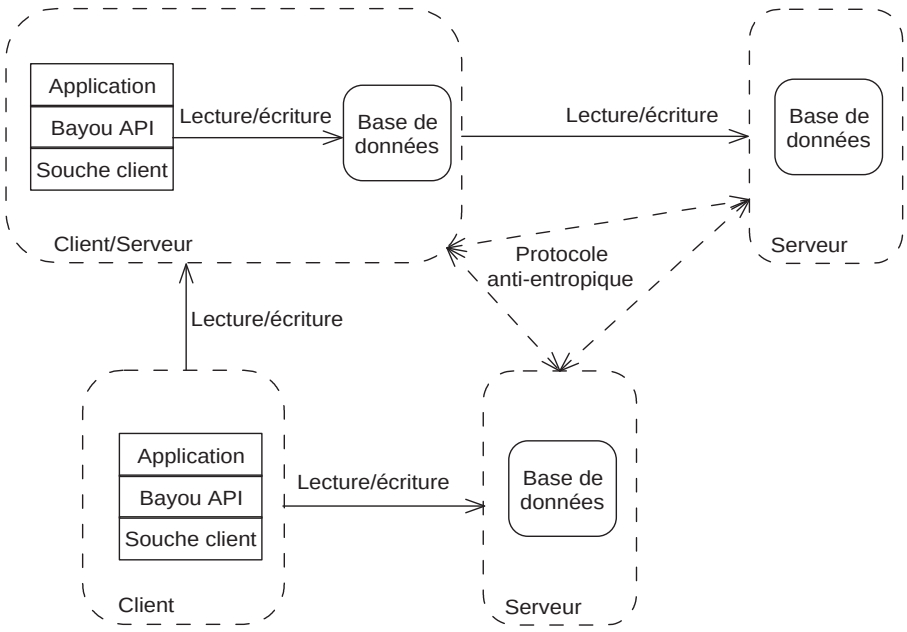

Figure 4: Architecture de Bayou.
Architecture of Bayou.

VI.2 Autres projets

Dans Oasis [53], en mode déconnecté, le client sur le terminal mobile utilise la base de données déployée localement, et en mode connecté, il utilise un mécanisme de vote à base de quorum (en anglais, *weighted voting*) pour choisir le serveur le plus à jour. Ce mécanisme de vote est aussi utilisé dans Deno [8] pour la réconciliation. Chaque copie possède un poids; les transactions sont validées à la majorité et lorsqu'il n'existe pas de transaction en conflit pouvant rassembler un plus grand nombre de votes. Par conséquent, les journaux d'informations à propager contiennent les transactions validées, les transactions candidates et les votes.

VII Systèmes orientés page web

Très proches des systèmes orientés fichier, les systèmes orientés page web présentent beaucoup de propositions de stratégies de remplacement [3]. Toutefois, moins de travaux abordent explicitement la gestion des déconnexions. Nous décrivons dans cette section un prototype de cache web pour environnements mobiles: WebExpress, puis résumons les principaux algorithmes de remplacement et mentionnons quelques travaux sur le préchargement.

VII.1 WebExpress

WebExpress [22] met en œuvre le modèle client/intercepteur/serveur comme illustré dans la figure 5. Les intercepteurs côté client CSI (*Client Side Interceptor*) et côté serveur SSI (*Server Side Interceptor*) englobent tous les deux un cache de pages web et font office de serveurs web mandataires. Les connexions entre les CSI et les SSI sont émulées pour diminuer le nombre d'ouvertures de connexions `TCP` et la quantité de données transmises (les en-têtes HTTP ne sont pas répétés dans les requêtes et les réponses successives). Dans les modes connecté et partiellement connecté, une requête est servie, dans l'ordre, par le CSI, le SSI, puis le serveur d'origine. La première entité possédant la page web la fournit, sinon en garde une copie lors du passage de la réponse. En mode déconnecté, toutes les requêtes du navigateur sont exécutées par le CSI, celui-ci renvoyant une erreur s'il ne dispose pas d'une copie en cache. Pour les scripts CGI, WebExpress définit le mécanisme de *différenciation* pour garder en cache les paramètres des requêtes et faire des comparaisons (`diff` à la Unix) entre la requête (côté client) et la réponse (côté serveur) courantes, et les requêtes et les réponses précédentes.

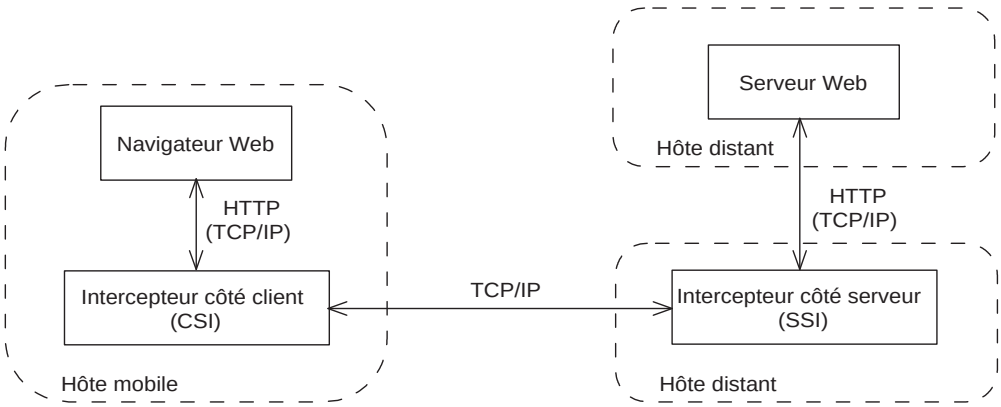

Figure 5: Architecture de WebExpress.
Architecture of WebExpress.

WebExpress associe à chaque page web dans le CSI ou le SSI un intervalle de cohérence qui indique la validité de la page web dans le cache. La valeur par défaut peut être modifiée par les utilisateurs au niveau de chaque CSI et par l'administrateur au niveau du SSI. Quand une page web dans le CSI (respectivement SSI) est référencée, si l'intervalle de cohérence est dépassé, le CSI (respectivement SSI) renvoie la requête vers le SSI (respectivement serveur web) et met à jour son cache au retour du résultat. Pour le remplacement, WebExpress utilise l'algorithme LRU dans le CSI et le SSI. Dans le CSI, l'utilisateur peut choisir un ensemble de pages web appelées *pages web infiniment persistantes*, qui sont exclues du remplacement. Par ailleurs, le déploiement d'une page

web dans le cache correspond au déploiement du code HTML de la page web et de toutes les images incluses dans la page web. Cependant, WebExpress ne prend pas en compte les liens hypertextuels qui pointent vers d'autres pages web. Donc, WebExpress ne prend pas en compte les dépendances entres les pages web. Contrairement à WebExpress, [37] pré-charge systématiquement la totalité des liens directs associés à la page web courante.

VII.2 Stratégies de remplacement

Dans SliCache [30], les paramètres des stratégies de remplacement sont le ratio entre la latence et la durée de chargement en fonction de la taille de la page; cette estimation est effectuée par le client. Le cache est ensuite divisé en deux parties: *R-slice* (*recency-slice*) contient des objets accédés une seule fois avec leur date d'accès, et *I-slice* (*interaccess-slice*) contient des objets accédés plusieurs fois avec les dates de leur dernier et avant-dernier accès. Les objets peuvent passer de *R-slice* à *I-slice*, pas l'inverse. Chaque partition utilise une fonction de coût: l'inverse de la taille de la page pour *R-slice* et le produit de l'intervalle d'accès avec le nombre d'accès pour *I-slice*. Ainsi, SliCache définit *R-victim* (respectivement *I-victim*) comme étant la page à moindre coût à remplacer dans *R-slice* (respectivement *I-slice*). Pour sélectionner la page web à remplacer (*R-victim* ou *I-victim*), SliCache utilise un mécanisme de prédiction: choix de la *I-victim* s'il y a une éventuelle référence de la *R-victim* avec une dégradation de la popularité de la *I-victim*, choix de la *R-victim* dans le cas contraire.

SIZE [66] remplace le document le plus volumineux par plusieurs documents moins volumineux, les documents de petite taille pouvant donc exister en permanence dans le cache sans être utilisés.

GDSF (*Greedy Dual Size with Frequency*) [10] assigne pour chaque document D une clé augmentant avec la fréquence d'accès, mais aussi avec la taille de l'entité et la durée estimée de déploiement de l'entité. GDSF remplace le document avec la clé la plus faible. Ainsi, les petits documents utilisés moins fréquemment sont plus sujets à éviction du cache.

Ariadne [56, 55] propose une politique de remplacement prenant en compte la consommation de l'énergie. Le poids d'une page est fonction de la popularité de la page (probabilité d'être prochainement accédée), du coût de chargement (à travers le réseau ad hoc si la page n'est pas dans le cache) et de la validité de la page (chaque page possède une durée de validité), elle-même fonction de l'énergie du terminal encore disponible (lorsque l'énergie est faible, il faut éviter les accès réseaux très gourmands en énergie).

VII.3 Pré-chargement

L'intérêt du pré-chargement de pages web a été montré en terme de réduction de la latence observée par l'utilisateur: [15] apporte un gain de 23%, [48] un gain de 45% (mais au prix d'un doublement de la bande passante utilisée). [26] étudie le pré-chargement dans le cas

spécifique de la tolérance aux déconnexions. Les auteurs décomposent leur architecture en deux modules. Premièrement, le module de prédiction calcule les probabilités d'accès futurs (proches) aux pages accessibles à partir des pages actuellement utilisées. Cette évaluation est effectuée par le client (apprentissage personnel) et par le serveur (apprentissage collectif d'une communauté de clients). Lorsque le client accède régulièrement à une page, c'est le calcul fait par le client qui est gardé, et lorsque la page est peu accédée par le client alors c'est la mesure (collective) du serveur qui est choisie. Deuxièmement, le module de pré-chargement prend les décisions de chargement en fonction de trois critères: la durée de pré-chargement de la page, la consommation de bande passante pour charger la page et l'impact sur les autres clients lors de l'éventuel chargement. Pour les déconnexions involontaires, ces mesures sont adaptées aux conditions réseau. Les auteurs ajoutent en plus un facteur coût de communication: coût d'accès au réseau plus coût de transfert de données. Pour les déconnexions volontaires, le module de pré-chargement demande à l'utilisateur de lui indiquer les pages de départ de navigation. Ces pages se voient attribuer la probabilité 1. Le module de prédiction calcule les probabilités des pages accessibles à partir de ces premières pages et remplit un cache dédié. Dans [9], pour les utilisateurs nomades, les auteurs associent la prédiction de déplacement avec la prédiction de navigation sur le web: si quelqu'un fait le même trajet que vous, il faut proactivement vous proposer les pages accédées par l'autre utilisateur pendant son trajet.

VIII Systèmes orientés objet

Avec les systèmes orientés objet est apparue la modélisation orientée objet. La grande nouveauté est l'apparition des premières méthodes de conception d'applications réparties tolérant les déconnexions. Grâce à l'encapsulation, un objet regroupe des données et le code pour manipuler ces dernières. Ensuite, grâce au polymorphisme, les objets déconnectés présents sur le terminal mobile peuvent être construits en surchargeant les objets serveurs: il est donc possible de séparer en partie la gestion des déconnexions du code métier. Dans cette section, nous décrivons les projets relatifs à la gestion du cache pour les applications orientées objet. Dans un premier temps, nous présentons Rover et CASCADE. Ensuite, nous résumons les résultats de quelques autres travaux de recherche.

VIII.1 Rover

Rover [29, 28] propose un modèle de programmation basé sur deux concepts: objet dynamique relogeable RDO (*Relocatable Dynamic Object*) et appel de procédure à distance non bloquant QRPC (*Queued Remote Procedure Call*). Un RDO est un objet qui peut être dynamiquement chargé (répliqué) sur le client à partir du serveur. Les QRPC constituent un système de communication qui permet aux applications de continuer à faire des RPC non bloquants lorsque la machine est déconnectée. Les applications sont construites en

utilisant des RDO et des QRPC.

L'architecture de Rover est structurée en trois couches: application, support système et transport, et composée de quatre entités (cf. figure 6). Le cache d'objets se compose d'un cache local privé situé dans l'espace d'adressage de l'application, et d'un cache local partagé par les applications co-localisées sur le même terminal qui est situé dans l'espace d'adressage du gestionnaire d'accès. Le gestionnaire du réseau met en œuvre un mécanisme de contrôle de la bande passante et optimise la transmission du journal en regroupant les opérations destinées au même serveur.

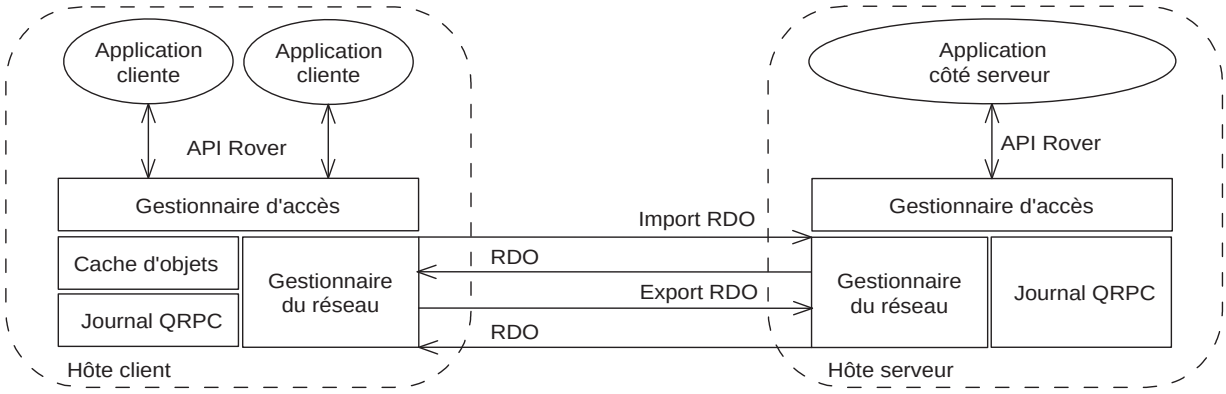

Figure 6: Architecture de Rover.
Architecture of Rover.

En mode connecté, lorsque le client invoque une méthode d'un objet, le gestionnaire d'accès contrôle d'abord si l'objet réside dans le cache. Si c'est le cas, Rover applique l'invocation directement sur l'objet du cache alors marqué *provisoirement valide*. Il est marqué *valide* lorsque l'invocation est transmise à l'objet serveur. Si l'objet est non présent dans le cache, Rover importe d'abord le RDO correspondant. En modes déconnecté et partiellement connecté, si le RDO n'est pas dans le cache, toutes les communications étant asynchrones, Rover enregistre un QRPC dans le journal d'opérations et retourne le contrôle à l'application.

Rover propose deux stratégies de déploiement: la première stratégie consiste à donner à l'utilisateur la possibilité de choisir les objets à déployer au lancement de l'application via une interface graphique. La deuxième stratégie consiste à déployer les objets à la première invocation. Rover laisse la liberté de prendre d'autres paramètres dans le choix des objets à déployer. Par exemple, dans l'application de messagerie électronique Exmh de Rover [28], la stratégie de déploiement prend en considération la taille des messages. Rover expose cette information à l'utilisateur afin de l'aider à choisir les objets à déployer. Par contre, Rover ne propose aucune stratégie de remplacement.

VIII.2 CASCADE

CASCADE [11] propose un service de gestion répartie de cache pour objets CORBA. Ce service est fourni par plusieurs serveurs de cache appelés DCS (*Domain Caching Servers*). Chaque DCS est responsable d'un domaine logique qui peut correspondre à une zone administrative comme dans le DNS. Le déploiement des copies dans les DCS se fait à la première invocation. Le service de gestion du cache de CASCADE n'est pas destiné à la gestion des déconnexions. Néanmoins, il propose des stratégies de déploiement et de remplacement de cache. La stratégie de remplacement utilise deux paramètres: le nombre maximum d'objets à déployer sur chaque DCS et la taille maximale de chaque objet présent dans chaque DCS. Lorsque le nombre d'objets dans un DCS atteint la limite haute, le DCS exécute l'algorithme LRU pour supprimer des objets déjà déployés jusqu'à ce que l'espace mémoire devienne suffisant pour les nouveaux objets. Par ailleurs, tout objet dans le DCS dépassant sa taille maximale sera supprimé. [2] propose deux autres algorithmes de remplacement: *H-BASED* et *LFU-H-BASED*. *H-BASED* définit pour chaque objet dans le cache la méta-donnée *clé* qui représente le nombre de fois où l'objet a été supprimé pour le remplacement; l'objet avec la plus petite clé est choisi pour le remplacement. En plus de la clé, *LFU-H-BASED* ajoute la méta-donnée *priorité*. La priorité est dynamique et calculée par CASCADE[4]. Lorsqu'un objet doit être supprimé du cache, *LFU-H-BASED* choisit l'objet le moins prioritaire. Si plusieurs objets sont moins prioritaires, l'algorithme *H-BASED* est appliqué.

La structure des copies secondaires est différente de la structure de la copie originale. En effet, chaque copie secondaire est encapsulée dans un module qui comporte en plus de l'objet CORBA, des méta-données et un contrôleur de requêtes. Les méta-données sont utilisées pour décrire la copie (localisation, DCS racine, etc.). Le contrôleur des requêtes reçoit toutes les requêtes destinées à l'objet pour ajouter des traitements extrafonctionnels (persistance, sécurité, etc.) avant de les rediriger vers l'objet CORBA. En outre, CASCADE utilise des intercepteurs au niveau client pour communiquer avec le DCS local (création de la copie et redirection des requêtes) d'une manière transparente.

IX Systèmes orientés composant

Un composant est une entité logicielle qui fournit et requiert des services, regroupés au sein d'interfaces. Les composants sont assemblés à l'aide de liaisons représentant des chemins de communication entre une interface requise par un composant et une interface compatible fournie par un autre composant. L'orientation composant est souvent associée avec un langage de description d'architecture.

L'orientation composants étant plus récente, moins de solutions de gestion des déconnexions pour applications à base de composants existent dans la littérature. C'est

[4]Les auteurs ne précisent pas comment CASCADE calcule la priorité.

aussi le modèle de conception le plus puissant en terme d'expressivité des politiques de gestion du cache. En effet, les modèles de composants rendent explicites les dépendances entre composants via les interfaces offertes et requises. En outre, un composant est par définition une entité de déploiement [63], contrairement aux objets par exemple qui sont à grain généralement plus fin. Dans cette section, nous présentons DisconnectP, puis MADA/DOMINT, avant de terminer par une brève description du projet Achille.

IX.1 DisconnectP

[41] propose une méthodologie de configuration non intrusive de la duplication et de la cohérence en se basant sur la séparation entre les préoccupations fonctionnelles et les préoccupations extrafonctionnelles des composants. Les protocoles sont modélisés en composants indépendants des applications métier. Dans le domaine d'utilisation de la duplication, DisconnectP est un protocole de gestion des déconnexions volontaires pour applications à base de composants. Dans ce protocole, chaque composant client est surchargé par des interfaces requises et un composant serveur par des interfaces offertes. Ces interfaces permettent de définir de façon générique les interactions entre les composants de l'application et les services de l'intergiciel. Même si DisconnectP ne propose pas de méthodologie de conception, ces interfaces sont un premier pas pour en définir une.

La stratégie de déploiement utilisée dans *DisconnectP* est manuelle, à la demande de l'utilisateur. Le gestionnaire du cache ne propose ni stratégie de remplacement ni gestion de dépendances entre les composants de l'application.

IX.2 MADA/DOMINT

MADA (*Mobile Application Development Approach*) est une méthodologie de conception d'applications pouvant fonctionner en présence de déconnexions, centrée sur l'architecture logicielle et pilotée par les cas d'utilisation (c'est-à-dire, partant des fonctionnalités fournies aux utilisateurs finaux) [34, 36]. L'approche MADA suit l'approche MDA (*Model Driven Architecture*) [46] de l'OMG. L'architecture logicielle de l'application est surchargée de spécifications dédiées à la gestion de déconnexion. Cette modélisation UML est indépendante de la plateforme d'implantation et d'exécution. Elle est ensuite transformée en un modèle ayant pour cible la plate-forme composant CORBA OpenCCM[5], tout en utilisant la plate-forme DOMINT (présentée ci-dessous).

Dans MADA, une application est vue comme un assemblage de composants qui interagissent entre eux pour réaliser les fonctionnalités de l'application. Ces fonctionnalités sont vues comme des services de l'application interagissant entre eux. MADA explicite donc des dépendances entre composants (vision « grain fin » de l'application manipulée par l'architecte et les développeurs), entre services et composants, et entre services (vision

[5]http://openccm.objectweb.org

« gros grain » de l'application manipulée par l'utilisateur). L'architecte de l'application assigne pour chaque composant (service) de l'application trois méta-données: *déconnectabilité*, *nécessité* et *priorité*. La déconnectabilité indique si un composant (service) de l'application peut avoir un mandataire sur le terminal mobile. La nécessité indique si la présence du mandataire, appelé composant (service) déconnecté dans le cache du terminal mobile est absolument nécessaire pour le fonctionnement de l'application en présence des déconnexions. La priorité permet d'ordonner les composants (services) de l'application quant à leur présence dans le cache du terminal mobile. La déconnectabilité est uniquement attribuée par l'architecte de l'application : c'est l'architecte qui interdit, par exemple, qu'un composant soit décliné en composant déconnecté sur un terminal mobile pour raison de sécurité. L'utilisateur quant à lui, peut surcharger dynamiquement la nécessité et la priorité des services fixées initialement par l'architecte. Enfin, des règles pour le déploiement et le remplacement définissent la propagation de la nécessité dans le graphe des dépendances entre services et composants.

La plateforme supportant MADA s'appelle DOMINT (*Disconnected Operation for Mobile INternetworking Terminals*) [34, 35], basé sur l'intergiciel OpenCCM. Les déconnexions peuvent être volontaires et involontaires. L'architecture de DOMINT est découpée en trois couches: composant, conteneur et intergiciel (cf. figure 7). La couche composant constitue la partie métier de l'application. Un composant peut offrir des interfaces et utiliser d'autres interfaces offertes par d'autres composants. De ce fait, le composant peut être un client pour un ensemble de composants et un serveur pour d'autres. Le composant serveur sur la figure 7 peut donc être un composant client pour un autre composant serveur.

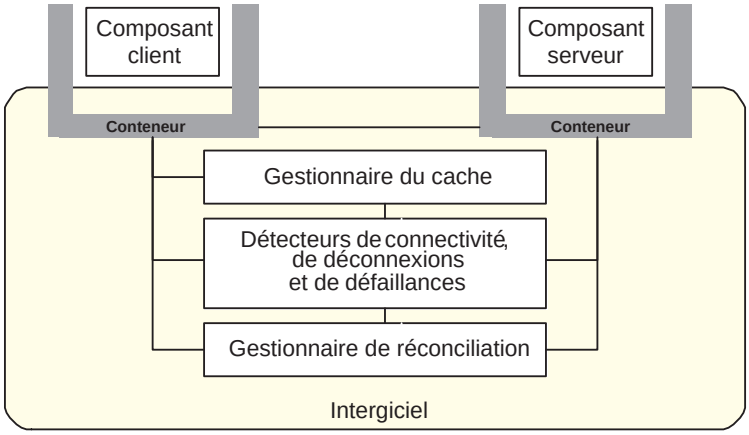

Figure 7: Architecture de DOMINT.
Architecture of DOMINT.

La couche intergiciel offre trois services: le gestionnaire du cache, le gestionnaire de

réconciliation, et les détecteurs de connectivité, de déconnexions et de défaillances. Plus particulièrement pour notre étude, le gestionnaire du cache centralise la gestion de tous les composants déconnectés dans le cache du terminal mobile. Il fournit la mise en œuvre des stratégies de déploiement et de remplacement du cache basées sur le profil de l'application et le graphe de dépendances de MADA. La stratégie de déploiement se décline en trois étapes complémentaires: au déploiement initial (pré-chargement), à la demande (explicite) de l'utilisateur et à l'invocation (implicite). La stratégie de remplacement se décline elle aussi en trois étapes: à la demande de l'utilisateur, en cas de manque de place, et de façon périodique pour préserver une zone libre dite « critique » afin d'accélérer de possibles déploiements urgents.

L'environnement d'exécution d'un composant CORBA est contrôlé par le conteneur. Ce dernier héberge les instances des composants et fournit des interfaces pour l'activation et la désactivation des serviteurs CORBA, l'accès à l'ORB sous-jacent, etc. Dans DOMINT, le service de gestion des déconnexions est contrôlé par le conteneur. Il coordonne la commutation transparente entre le composant serveur et le composant déconnecté suivant le mode de fonctionnement. Il est aussi responsable de l'orchestration des différents services de la couche intergiciel de DOMINT dans le but de séparer les aspects fonctionnels de l'application de l'aspect gestion des déconnexions.

IX.3 Achilles

Achilles [33] est un canevas logiciel pour la gestion des déconnexions volontaires. Achilles propose deux stratégies de déploiement: à la demande de l'utilisateur et à l'invocation. Dans la première stratégie, l'utilisateur peut indiquer des paquetages permanents qui doivent toujours être présents dans le cache. Achilles contrôle la *qualification* du paquetage, c'est-à-dire l'ensemble des ressources (matérielles ou logicielles) dont le terminal mobile doit disposer pour que le paquetage puisse être utilisé localement. Ces relations de dépendances sont modélisées dans un graphe. Achilles propose aussi deux stratégies de remplacement: à la demande de l'utilisateur et automatique. Aucune des deux ne garantit le maintien des dépendances entre paquetages. La deuxième stratégie exécute un algorithme de coût minimum. Le coût d'un paquetage prend en considération deux paramètres: le coût de redéploiement du paquetage une fois supprimé, et l'importance du paquetage. Le coût de redéploiement est fonction de la taille du paquetage et de la bande passante disponible. L'importance d'un paquetage dans le cache est calculé comme étant le nombre de paquetages qui en dépendent.

X Synthèse générale et perspectives

La continuité de service entre le client et les serveurs en présence des déconnexions est assurée par le maintien d'une connexion logique en utilisant le concept d'opération décon-

nectée. Grâce à ce concept, nous avons identifié quatre intervenants: le client sur l'hôte mobile, l'entité distante de l'application répartie, le cache qui représente une partie de la mémoire du terminal mobile et l'entité déconnectée agissant comme un mandataire en termes de code et d'état de l'entité distante dans le cache. Nous avons ensuite présenté le fonctionnement des opérations déconnectées. Ce fonctionnement fait apparaître quatre mécanismes de base: la gestion du cache du terminal mobile, la gestion de la cohérence des différentes entités, la détection des déconnexions, et la commutation entre l'entité distante et l'entité déconnectée. L'état de l'art que nous présentons dans cet article aborde uniquement les thématiques de gestion du cache, et de commutation entre l'entité distante et l'entité déconnectée.

Nous avons étudié les besoins en terme d'adaptation aux déconnexions des terminaux mobiles. Cette étude identifie quatre caractéristiques qui définissent l'adaptation. Tout d'abord, la stratégie d'adaptation définit les acteurs de l'adaptation. Nous avons énuméré trois stratégies: *laisser-faire*, *transparence* et *collaboration*, cette dernière étant à préférer pour les applications réparties devant fonctionner en environnements mobiles. Par ailleurs, le type d'adaptation définit le moment de l'adaptation. Celle-ci peut être statique, dynamique ou auto-adaptable. Ensuite, les politiques d'adaptation expriment les mécanismes à mettre en œuvre. Enfin, les mécanismes d'adaptation opérent les actions d'adaptation. Cet article a présenté plus précisément un état de l'art de la gestion du cache logiciel pour tolérer les déconnexions en environnements mobiles. Cette analyse montre que, pour l'adaptation aux déconnexions des terminaux mobiles, il est souhaitable de mettre en œuvre un mixage entre les différents types d'adaptation, et que cette adaptation doit être effectuée suivant la stratégie collaboration, en raison de la non-adéquation des stratégies laisser-faire et transparence.

Le tableau I récapitule des différentes solutions étudiées pour la gestion des déconnexions en environnements mobiles, au vu des différents critères. La deuxième colonne du tableau représentant le premier critère permet de classer les solutions selon le critère le plus discriminant : la granularité ou orientation du système. Nous retrouvons alors classiquement les évolutions historiques du domaine du génie logiciel. Les premières solutions apparaissent dans les systèmes d'exploitation pour modifier le système de gestion répartie de fichiers. Même s'ils apparaissent beaucoup plus tard, les systèmes orientés page web introduisent les liens hypertextuels qui sont une manière d'expliciter des relations de dépendances entre fichiers. Puis, c'est la structure interne des fichiers qui est décomposée logiquement en tables, enregistrements et champs dans les bases de données. Les solutions profitent alors des concepts de relation entre tables et de transaction pour délimiter des portions de code exécutées isolément et de façon atomique. La gestion du cache et de la réconciliation devient ainsi plus fine. Ensuite, l'orientation objets survient de façon concomitante avec l'essor des méthodes de conception, donnant naissance aux premières méthodes de conception d'applications réparties tolérant les déconnexions. En outre, les propriétés des objets, principalement l'encapsulation (du code et

Solution	Granularité	Type de déconnexion	Mode de fonctionnement	Type et stratégie d'adaptation	Stratégie de déploiement	Stratégie de remplacement	Modélisation lors de la conception	Traitement des dépendances
Coda	Fichier	Volontaire et involontaire	C, P et D	Statique, dynamique et collaboration	Pré-chargement et à la demande	LRU	Profil utilisateur, données implicites et explicites	
Seer/ Roam	Fichier	Volontaire et involontaire	C et D	Dynamique et transparence	Prédiction		Projet et distance sémantique	Inférence par observation: intra-projet
Amigos	Fichier	Volontaire et involontaire	C et D	Statique, dynamique et collaboration	Pré-chargement	À la demande	Profil utilisateur	
Bayou	Base de données	Volontaire et involontaire	C, P et D	Statique, dynamique et collaboration	À la demande		*Read-any/Write-any*, opération R/W et protocole anti-entropique	Inférence par observation: horloges logiques vectorielles
WebExpress	Page Web	Involontaire	C, P et D	Statique, transparence, dynamique et collaboration	À l'invocation	LRU	Serveur web local, intervalle de cohérence et pages web infiniment persistantes	
SliCache	Page Web	Involontaire	C et D	Statique et transparence		Prédiction	Partitionnement du cache (*R-slice* et *I-slice*)	
Rover	Objet	Volontaire et involontaire	C, P et D	Statique, dynamique et collaboration	Pré-chargement et à l'invocation		RDO, QRPC et canevas logiciel	
CASCADE	Objet		C	Statique, dynamique et collaboration	À l'invocation	LRU, H-BASED et LFU-H-BASED	Structuration de l'application cliente	
DisconnectP	Composant	Volontaire et involontaire	C et D	Statique, dynamique et collaboration	À la demande		Protocole de gestion des déconnexions	
MADA / DOMINT	Composant	Volontaire et involontaire	C, P et D	Statique, dynamique, auto-adaptation et collaboration	Pré-chargement, à la demande, à l'invocation	À la demande, en cas de manque de place, périodique, LFUPP	Approche MDA, centrée architecture, basée sur les cas d'utilisation, avec méta-données	Explicitation des dépendances entre composants et services

Table I: Récapitulatif des principales solutions de la littérature. Les abréviations sont les suivantes: C pour connecté, P pour partiellement connecté et D pour déconnecté; les cases vides correspondent à une information non connue.
Synthesis of the main solutions of the litterature. The abbreviations have the following meaning: C for connected, P for partially connected, and D for disconnected; the empty cells correspond to unknown information.

des données) et le polymorphisme, permettent de commencer à concevoir des solutions séparant en partie la gestion des déconnexions du code métier de l'application répartie. Enfin, le modèle de conception orienté composant est à ce jour le plus puissant en terme d'expressivité d'adaptation pour la gestion du cache. En effet, les modèles de composants définissent un composant comme une unité de déploiement et rendent explicites les dépendances entre composants via les interfaces offertes et requises ; les travaux sur les stratégies de déploiement et de remplacement du cache abondamment étudiées dans les autres orientations sont alors applicables. Ces stratégies bénéficient en plus des mécanismes de réflexivité pour introduire et manipuler des méta-données ; ces méta-données permettent une connaissance du contexte: environnement d'exécution, profil architectural de l'application, profil de l'utilisateur final... ; cette connaissance du contexte autorise par conséquent l'exploration de nouvelles stratégies de déploiement et de remplacement. Toutes ces avancées amènent de plus en plus à considérer l'adaptation comme un aspect à tisser statiquement ou dynamiquement pendant l'exécution de l'application [13, 31].

En conclusion, la dimension « gestion des déconnexions » doit être traitée en collaboration entre l'application et le système. Malheureusement, hormis les approches orientées composant, la plupart des solutions existantes ne proposent pas de séparation entre les préoccupations fonctionnelles de l'application et la gestion des déconnexions. Cette contrainte limite les possibilités de maintenance, de réutilisation et de reconfiguration. Ces mêmes solutions ne proposent pas non plus de modèle de conception. Une autre limitation souvent présente est la non prise en compte des dépendances entre entités dans les stratégies de déploiement et de remplacement. Ces limitations sont conceptuellement en partie levées avec le paradigme composant, mais pas encore suffisamment étudiées. C'est pourquoi nous pensons que les progrès à venir sont à attendre, d'une part, d'un meilleur tissage des services extrafonctionnels sur les composants de l'application (plan de base) via des contrôleurs de composants (plan méta), et d'autre part, d'une meilleure expression des dépendances en termes de fonctionnalités (via par exemple les interfaces offertes et requises) et de données (via par exemple les méta-données distinguant les composants *entité*, *session*... comme dans les modèles de composants OMG CCM [47] et Sun EJB [62] ou les méta-données exprimant les dépendances comme dans les bases de données).

La deuxième perspective que nous voyons aux travaux existants est le partage de caches logiciels pour former des caches répartis, tout en prenant en compte la forte dynamicité des environnements mobiles, voire des environnements plus dynamiques comme ceux des réseaux spontanés (ad hoc) dans lesquels les communications possèdent aussi la particularité d'être pair à pair. Dans les réseaux spontanés, en plus des déconnexions, des partitionnements surviennent suite aux ruptures des communications avec les stations de base des infrastructures des réseaux sans fil cellulaires. Cependant, les communications pair-à-pair sont toujours possibles, d'où l'idée de caches collaboratifs [7, 12, 42, 52, 56, 67]. Dans ces environnements, les terminaux mobiles sont rassemblés dans des groupes. Ces groupes sont soit pré-existants et calqués sur la topologie des différents réseaux [5, 14],

soit spécialement formés par la gestion collaborative du cache [12, 52]. Les recherches sur ce sujet utilisent aussi comme point de départ les propriétés des algorithmes de routage dans les réseaux spontanés [56]. Une autre direction de recherche est le support de la mobilité des terminaux qui passent d'un réseau spontané à un autre [7, 67]. Dans les réseaux pair-à-pair à large échelle, la problématique est de contraindre le réseau de pairs à une zone administrative, et de partager l'espace de cache et d'accès longue distance, par exemple comme dans [25, 42], l'intérêt étant ici d'appliquer ces principes à de petits réseaux comme les réseaux spontanés.

Remerciements

Les auteurs remercient les membres de l'équipe Marge du laboratoire CNRS Samovar pour leurs relectures des différentes versions de l'article, et plus spécialement, Sophie Chabridon, Mejdi Kaddour, Michel Simatic et Chantal Taconet. Sont aussi ici remerciés Laurence Duchien, Areski Flissi et Romain Rouvoy pour leur relecture de la première version de l'article. Bien entendu, la version finale de l'article a bénéficié des nombreux commentaires des rapporteurs anonymes. Qu'ils en soient ici vivement remerciés.

References

[1] Andersen (B.), Jul (E.), Moura (F.), Guedes (V.M.P.). File System for Semiconnected Operation in AMIGOS. *Proc. 2nd USENIX Symposium on Mobile and Location-Independent Computing*, décembre 1994.

[2] Atzman (H.), Friedman (R.), Vitenberg (R.). Replacement Policies for a Distributed Object Caching Service. R. Meersman, Z. Tari (éditeurs), *Proc. 4th International Symposium on Distributed Objects and Applications*, *Lecture Notes in Computer Science*, volume 2519. Springer-Verlag, University of California at Irvine, USA, p. 661–674, octobre 2002.

[3] Barish (G.), Obraczke (K.). World Wide Web Caching: Trends and Techniques. *IEEE Communications Magazine*, **38(5)**:p. 178–184, May 2000.

[4] Bernard (G.), Ben-Othman (J.), Bouganim (L.), Canals (G.), Chabridon (S.), Defude (B.), Ferrié (J.), Gançarski (S.), Guerraoui (R.), Molli (P.), Pucheral (P.), Roncancio (C.), Serrano-Alvarado (P.), Valduriez (P.). Mobile databases: a selection of open issues and research directions. *ACM SIGMOD Record*, **33(2)**:p. 78–83, June 2004.

[5] Boulkenafed (M.), Issarny (V.). A middleware Service for Mobile Ad Hoc Data Sharing, Enhancing Data Availability. M. Endler, D. Schmidt (éditeurs),

Proc. IFIP/ACM/USENIX International Middleware Conference, *Lecture Notes in Computer Science*, volume 2672. Springer-Verlag, Rio de Janeiro, Brazil, p. 493–511, juin 2003.

[6] Bruneton (É.). *Un support d'exécution pour l'adaptation des aspects non-fonctionnels des applications réparties*. Thèse de doctorat, INPG, Grenoble, France, 2001.

[7] Cao (G.), Tin (L.), Das (C.R.). Cooperative Cache-Based Data Access in Ad Hoc Networks. *IEEE Computer*, **37(2)**:p. 32–39, février 2004.

[8] Çetintemel (U.), Keleher (P.J.), Bhattacharjee (B.), Franklin (M.J.). Deno: A Decentralized, Peer-to-Peer Object-Replication System for Weekly Connected Environments. *IEEE Transactions on Computers*, **52(7)**:p. 943–959, juillet 2003.

[9] Chelcea (S.), Gallais (G.), Trousse (B.). Recommandations personnalisées pour la recherche d'information facilitant les déplacements. *Actes de la 1ère Conférence ACM Francophone Mobilité et Ubiquité*. Nice, France, p. 143–150, juin 2004.

[10] Cherkasova (L.). Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy. Rapport technique, HP Labs, Palo Alto, November 1998.

[11] Chockler (G.V.), Dolev (D.), Friedman (R.), Vitenberg (R.). Implementing a Caching Service for Distributed CORBA Objects. J. Sventek, G. Coulson (éditeurs), *Proc. IFIP/ACM/USENIX International Middleware Conference*, *Lecture Notes in Computer Science*, volume 1795. Springer-Verlag, p. 1–23, avril 2000.

[12] Chow (D.-Y.), Leong (H.V.), Chan (A.T.S.). Group-based Cooperative Cache Management for mobile Clients in a Mobile Envrionement. *Proc. International Conference on Parallel Processing*. p. 83–90, août 2004.

[13] David (P.-C.). *Développement de composants Fractal adaptatifs: un langage dédié à l'aspect d'adaptation*. Thèse de doctorat, Université de Nantes, École des Mines de Nantes (France), juillet 2005.

[14] Dwyer (D.), Bharghavan (V.). A mobility-aware file system for partially connected operation. *ACM Operating Systems Review*, **31(1)**:p. 24–30, janvier 1997.

[15] Fan (L.), Cao (P.), Lin (W.), Jacobson (Q.). Web prefetching between low-bandwidth clients and proxies: potential and performance. *ACM SIGMETRICS Performance Evaluation Review*, **27(1)**:p. 178–187, June 1999.

[16] Flinn (F.), Narayanan (D.), Satyanarayanan (S.). Self-Tuned Remote Execution for Pervasive Computing. *Proc. 8th Workshop on Hot Topics in Operating Systems*. Elmaü, Germany, p. 61–66, May 2001.

[17] Franklin (M.J.). Transactional Client-Server Cache Consistency: Alternatives and Performance. *ACM Transactions on Database Systems*, **22(3)**:p. 315–363, september 1997.

[18] Froese (K.W.), Bunt (R.B.). Cache Management for Mobile File Service. *The Computer Journal*, **42(6)**:p. 442–454, juin 1999.

[19] Garlan (D.), Siewiorek (D.P.), Smailagic (A.), Steenkiste (P.). Project Aura: Toward Distraction Free Pervasive Computing. *IEEE Pervasive Computing*, **1(2)**:p. 22–31, avril 2002.

[20] Gurun (S.), Krintz (C.), Wolski (R.). NWSLite: A Light-Weight Prediction Utility for Mobile Devices. *Proc. 2nd ACM/USENIX International Conference on Mobile Systems, Applications and Services*. Boston, USA, p. 2–11, juin 2004.

[21] Guy (R.G.), Heidemann (J.S.), Mak (W.-K.), Page (T.W.), Popek (G.J.), Rothmeier (D.). Implementation of the Ficus Replicated File System. *Proc. USENIX Technical Conference*. Anaheim, California (USA), p. 63–72, juin 1990.

[22] Housel (B.C.), Samaras (G.), Lindquist (D.B.). WebExpress: A client/intercept based system for optimizing Web browsing in a wireless environment. *ACM Mobile Networks and Applications*, **3(4)**:p. 419–431, décembre 1998.

[23] Howard (J.H.), Kazar (M.L.), Menees (S.), Nichols (D.), Satyanarayanan (M.), Sidebotham (R.N.), West (M.J.). Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, **6(1)**:p. 51–81, February 1988.

[24] Huston (L.B.), Honeyman (P.). Partially Connected Operation. *Proc. 2nd USENIX Symposium on Mobile and Location-Independent Computing*. Ann Arbor, Michigan, USA, p. 91–98, avril 1995.

[25] Iyer (S.), Rowstron (A.), Druschel (P.). Squirrel: A decentralized peer-to-peer web cache. *Proc. 21st ACM Symposium on Principles of Distributing Computing*. Monterey, California, USA, p. 213–222, juillet 2002.

[26] Jiang (Z.), Kleinrock (L.). Web Prefetching in a Mobile Environment. *IEEE Personal Communications*, **5(5)**:p. 25–34, octobre 1998.

[27] Jing (J.), Helal (A.), Elmagarmid (A.). Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, **31(2)**:p. 117–157, juin 1999.

[28] Joseph (A.D.), deLespinasse (A.F.), Tauber (J.A.), Gifford (D.K.), Kaashoek (M.F.). Rover: A Toolkit for Mobile Information Access. *Proc. 15th ACM Symposium on Operating Systems Principles*. p. 156–171, décembre 1995.

[29] Joseph (A.D.), Tauber (J.A.), Kaashoek (M.F.). Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers*, **46(3)**:p. 337–352, mars 1997.

[30] Katsaros (D.), Manopoulos (Y.). Web Caching in Broadcast Mobile Wireless Envrionments. *IEEE Internet Computing*, **8(3)**:p. 37–44, mai 2004.

[31] Kiczales (G.), Lamping (J.), Mendhekar (A.), Maeda (C.), Lopes (C.V.), Loingtier (J.-M.), Irwin (J.). Aspect-Oriented Programming. M. Aksit, S. Matsuoka (éditeurs), *Proc. 11th European Conference on Object-Oriented Programming*, *Lecture Notes in Computer Science*, volume 1241. Jyväskylä, Finnland, p. 220–242, juin 1997.

[32] Kistler (J.J.), Satyanarayanan (M.). Disconnected Operation in the Coda File System. *Proc. 13th ACM Symposium on Operating Systems Principles*. Pacific Grove, USA, p. 213–225, octobre 1991.

[33] Kortuem (G.), Fickas (S.), Segall (Z.). On-Demand Delivery of Software in Mobile Environments. *Proc. IPPS Workshop on Nomadic Computing*. Geneva, Switzerland, avril 1997.

[34] Kouici (N.). *Gestion des déconnexions pour application réparties à base de composants en environnements mobiles*. Thèse de doctorat, Institut National des Télécommunications, en co-accréditation avec l'Université d'Evry Val d'Essonne, Évry (France), novembre 2005.

[35] Kouici (N.), Conan (D.), Bernard (G.). Caching Components for Disconnection Management in Mobile Envrionments. Z. Tari *et al* (éditeur), *Proc. 6th International Symposium on Distributed Objects and Applications*, *Lecture Notes in Computer Science*, volume 3291. Springer-Verlag, Agia Napa, Cyprus, p. 1322–1339, octobre 2004.

[36] Kouici (N.), Conan (D.), Bernard (G.). MADA : une approche pour le développement d'applications mobiles. *Actes de la 1ère Conférénce ACM Francophone Mobilité et Ubiquité*. Nice, France, p. 78–85, juin 2004.

[37] Kroeger (T.M.), Long (D.D.E.), Mogul (J.C.). Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. *The USENIX Symposium on Internet Technologies and Systems*. Monterey, California, USA, p. 13–22, décembre 1997.

[38] Kuenning (G.). Design of the SEER predictive caching schema. *Workshop on Mobile Computing Systems and Applications*. Santa Cruz, CA, USA, 1994.

[39] Kuenning (G.H.), Popek (G.J.). Automated Hoarding for Mobile Computers. *Proc. 16th ACM Symposium on Operating Systems Principles*. Saint Malo, France, p. 1–22, octobre 1997.

[40] Lynch (N.). *Supporting Disconnected Operation in Mobile CORBA*. Rapport de master, University of Dublin, Trinity College, septembre 1999.

[41] Marangozova (V.). *Duplication et cohérence configurables dans les applications réparties à base de composants*. Thèse de doctorat, Université Jospeh Fourier, Grenoble, France, juin 2003.

[42] Mislove (A.), Haeberlen (A.), Post (A.), Druschel (P.). ePOST. R. Steinmetz, K. Wehrle (éditeurs), *Peer-to-Peer Systems and Applications*, *Lecture Notes in Computer Science*, volume 3485, chapitre 13. Springer, p. 171–192, 2005.

[43] Mummert (L.B.), Ebling (M.R.), Satyanarayanan (M.). Exploiting Weak Connectivity for Mobile File Access. *Proc. 15th ACM Symposium on Operating Systems Principles*. p. 143–155, décembre 1995.

[44] Noble (B.D.), Satyanarayanan (M.). Experience with adaptive mobile applications in Odyssey. *ACM Mobile Networks and Applications*, **4(4)**:p. 245–254, 1999.

[45] Noble (B.D.), Satyanarayanan (M.), Narayanan (D.), Tilton (J.E.), Flinn (J.), Walker (K.R.). Agile Application-Aware Adaptation for Mobility. *Proc. 16th ACM Symposium on Operating Systems Principles*. Saint Malo, France, p. 276–287, octobre 1997.

[46] OMG. MDA Guide Version 1.0. omg/2003-05-01, Object Management Group, mai 2003.

[47] OMG. CORBA Components. OMG Document formal/02-06-65, Version 3.0, Object Management Group, June 2002.

[48] Padmanabhan (V.N.), Mogul (J.C.). Using predictive prefetching to improve World Wide Web latency. *ACM SIGCOMM Computer Communication Review*, **26(3)**:p. 22–36, July 1996.

[49] Pawlowski (B.), Juszczak (C.), Staubach (P.), Smith (C.), Lebel (D.), Hitz (D.). NFS Version 3 Design and Implementation. *Proc. Summer USENIX Technical Conference*. Boston, MA (USA), p. 137–152, juin 1994.

[50] Petersen (K.), Spreitzer (M.J.), Terry (D.B.), Theimer (M.M.), Demers (A.J.). Flexible Update Propagation for Weakly Consistent Replication. *Proc. 16th ACM Symposium on Operating Systems Principles*. Saint Malo, France, p. 288–301, octobre 1997.

[51] Pitoura (E.), Bhargava (B.). Building Information Systems for Mobile Environments. *Proc. 3rd International Conference on Information and Knowledge Management*. Gaithesburg, MD, p. 371–378, novembre 1994.

[52] Ratner (D.), Reiher (P.), Popek (G.J.), Kuenning (G.H.). Replication Requirements in Mobile Environments. *ACM Mobile Networks and Applications*, **6(6)**:p. 525–533, novembre 2001.

[53] Rodrig (M.), LaMarca (A.). Oasis: An Architecture for Simplified Data Management and Disconnected Operation. *Personal and Ubiquitous Computing Journal*, **9(2)**, March 2005.

[54] Rudenko (A.), Reiher (P.), Popek (G.J.), Kuenning (G.H.). The Remote Processing Framework for Portable Computer Power Saving. *ACM Symposium on Applied Computing*. San Antonio, Texas, USA, p. 365–372, February 1999.

[55] Sailhan (F.). *Localisation de ressources dans les reseaux ad hoc*. Thèse de doctorat, Université Pierre et Marie Curie, PARIS VI (France), juillet 2005.

[56] Sailhan (F.), Issarny (V.). Cooperative Caching in Ad Hoc Networks. *Proc. of the 4th International Conference on Mobile Data Management*, *Lecture Notes in Computer Science*, volume 2574. Springer-Verlag, Melbourne, Australia, p. 13–28, janvier 2003.

[57] Saito (Y.), Shapiro (M.). Optimistic Replication. *ACM Computing Surveys*, **37(1)**:p. 42–81, mars 2005.

[58] Satyanarayanan (M.). Fundamental Challenges in Mobile Computing. *Proc. 15th ACM Symposium on Principles of Distributing Computing*. Philadelphia, USA, p. 1–7, mai 1996.

[59] Satyanarayanan, (M.). The Evolution of Coda. *ACM Transactions on Computer Systems*, **20(2)**:p. 85–124, mai 2002.

[60] Satyanarayanan (M.). The Many Faces of Adapatation. *IEEE Pervasive Computing*:p. 4–5, juillet 2004.

[61] Sousa (J.P.), Garlan (D.). Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environment. *Proc. 3rd Working IEEE/IFIP Conference on Software Architecture*, volume 1 (3). Montreal (Canada), p. 29–43, juillet 2002.

[62] Sun Microsystems. Enterprise javaBeans Specification. Version 2.1, Final Release, novembre 2002.

[63] Szyperski (C.). *Component Software: Beyond Object-Oriented Programming, 2nd edition*. Addison-Wesley, 2002.

[64] Temal (L.), Conan (D.). Détections de défaillances, de connectivité et de déconnexions. *Actes de la 1ère Conférénce Francophone Mobilité et Ubiquité*. ACM International Conference Proceedings Series, Nice, France, p. 90–97, juin 2004.

[65] Terry (D.B.), Theimer (M.M.), Petersen (K.), Demers (A.J.). Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. *Proc. 15th ACM Symposium on Operating Systems Principles*. p. 172–183, décembre 1995.

[66] Williams (S.), Abrams (M.), Standridge (C.R.), Abdulla (G.), Fox (E.A.). Removal Policies in Network Caches for World-Wide Web Documents. *Proc. Conference Applications, Technologies, Architectures, and Protocols for Computer Communications*. Palo Alto, California, USA, p. 293–305, 1996.

[67] Yin (L.), Cao (G.). Supporting Cooperative Caching in Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, **5(1)**:p. 77–89, janvier 2006.