Gestion de déconnexion pour applications réparties à base de composants dans un environnement mobile

Nabil Kouici, Lydialle Chateigner, Denis Conan, Sophie Chabridon et Guy Bernard

GET/INT, CNRS UMR SAMOVAR, 9 rue Charles Fourier, 91011 Évry, France
{Nabil.Kouici, Lydialle.Chateigner, Denis.Conan, Sophie.Chabridon, Guy.Bernard}@int-evry.fr

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 restant connecté aux applications réparties et d'être indépendant 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 sur les environnements mobiles doivent s'adapter aux déconnexions. Dans cet article, nous décrivons une méthodologie et une plateforme pour le développement et l'exécution d'applications réparties mobiles à base de composants. Plus particulièrement, nous détaillons la gestion du cache de composants et la gestion de la réconciliation. Nous décrivons enfin un scénario de mise en œuvre dans le domaine de la gestion de catastrophe.

Mots-clés :
Intergiciel, mobilité, déconnexion, composant, adaptabilité.

1. Introduction

Depuis le début des années 90, la forte évolution des réseaux sans fil suscite un intérêt croissant pour l'informatique [6]. Nous sommes successivement passés des réseaux locaux à des réseaux à grande échelle, puis à des réseaux sans fil interconnectant des machines mobiles comme des téléphones portables ou des assistants personnels numériques. Ainsi, un nouveau paradigme est apparu connu sous le nom de « informatique mobile ». L'informatique mobile offre aux utilisateurs la capacité de pouvoir se déplacer tout en restant connecté aux applications réparties et d'être indépendant de la localisation géographique. Cependant, l'informatique mobile souffre de plusieurs limitations : les terminaux mobiles sont limités en terme de mémoire, de batterie et de bande passante. De plus, le terminal mobile est sujet à des déconnexions. Ces déconnexions ne doivent pas être vues comme des défaillances puisqu'elles sont une simple conséquence de la mobilité même des utilisateurs [24].

Les déconnexions peuvent être volontaires car décidées par l'utilisateur depuis son terminal mobile (par exemple, pour économiser l'énergie de la batterie), comme elles peuvent être involontaires (par exemple, lors du passage de l'utilisateur dans une zone d'ombre radio). Nous définissons quatre modes de fonctionnement des terminaux mobiles [24]. Dans le mode connecté, le terminal mobile dispose d'une connexion normale au réseau, à la manière d'une station classique. Dans le mode partiellement connecté, le mobile ne dispose que d'une communication à faible débit. Le mode veille est utilisé par les terminaux mobiles pour préserver les ressources énergétiques. Dans ce cas, la connexion réseau est maintenue, le terminal mobile n'envoie plus de requêtes, mais il peut recevoir et repasser en mode partiellement connecté. Enfin, dans le mode déconnecté, le mobile ne dispose plus de lien physique.

Comme tout aspect extrafonctionnel, la gestion des déconnexions doit offrir le maximum de transparence aux utilisateurs. Celle-ci passe par une adaptation aux changements de niveau de disponibilité des ressources. Cette adaptation peut être entièrement de la responsabilité de l'application (stratégie « laissez-faire »), de la responsabilité du système ou de l'intergiciel[1] (stratégie « transparence »), ou enfin, effectuée par la collaboration entre l'application et le système (stratégie « collaboration ») [25]. De nombreux travaux synthétisés dans [10] montrent que les approches « laissez-faire » et « transparence » ne sont pas adéquates. Aussi, dans nos travaux, nous adoptons la stratégie « collaboration ». Cependant, les travaux

[1] Intergiciel est la traduction proposée de *middleware*.

cités ci-dessus ont traité le problème des déconnexions d'une manière plutôt ad hoc et peu d'entre eux proposent un méthode de développement d'applications réparties devant fonctionner en présence des déconnexions.

La contribution des travaux présentés dans cet article est la description de nos expériences dans la proposition d'une approche pour le développement des applications réparties mobiles, et de DOMINT[2], une plateforme de gestion de déconnexion des applications réparties à base de composants CORBA. L'idée principale de notre approche est de créer sur le terminal mobile des « composants déconnectés », qui sont des mandataires des composants distants. Ils sont similaires en conception et en implantation aux composants distants restant sur le réseau fixe, mais spécifiquement construits pour faire face au fonctionnement en présence des déconnexions. Il est de la responsabilité du concepteur de l'application d'opérer un compromis entre une conception simple et une conception plus complexe qui s'adapte plus efficacement aux variations de connectivité. Les composants déconnectés sont utilisés pour assurer la continuité de service lors des déconnexions. Les opérations effectuées pendant les phases de déconnexion sont journalisées localement pour être envoyées lors de la reconnexion aux serveurs pour la réconciliation. La réconciliation est « optimiste », permettant ainsi à plusieurs utilisateurs de travailler sur une copie des données sans poser de verrous. Le contrôle de la divergence des copies met en œuvre des transformées opérationnelles [30].

Pour la gestion du cache de composants logiciels, nous proposons une méthodologie de conception mettant en œuvre un ensemble de méta-données pour la construction d'un profil de l'application répartie. Le profil définit les composants ou les fonctionnalités de l'application qui peuvent ou doivent être présents dans le cache logiciel du terminal mobile pour le fonctionnement en modes partiellement connecté et déconnecté. Cette méthodologie est illustrée par la conception et la réalisation d'un service de gestion de cache de composants CORBA dans le cadre de la plateforme DOMINT. Nous définissons aussi la notion de service comme étant un ensemble de composants qui coopèrent pour la réalisation d'une fonctionnalité de l'application. Par ailleurs, nous maintenons en cache des composants qui sont connectés à d'autres composants. Ainsi, la solution que nous proposons pour la gestion du cache du terminal mobile prend en compte les dépendances entre les composants et les services de l'application, modélisées dans un graphe de dépendances hiérarchique.

Pour la gestion de la réconciliation de composants logiciels, notre approche a abouti à la conception et la réalisation d'un nouveau service extrafonctionnel fourni par le conteneur de composants. Cette proposition consiste plus précisément à étendre l'intergiciel en y intégrant ce service de réconciliation construit sur mesure pour la synchronisation de composants CORBA. Ce service permet de gérer la cohérence d'applications réparties multi-composants. Enfin, la connectivité est prise en compte. Parmi les fonctionnalités de ce service, durant la phase de déconnexion, nous pouvons énumérer la journalisation des opérations effectuées sur le terminal mobile. À la reconnexion, l'estampillage puis le transfert d'état sont effectués grâce à des composants dédiés. L'intégration des opérations distantes via les transformées permet de gérer les éventuels divergences et conflits pour obtenir à terme un état cohérent.

La suite du papier est organisée comme suit. Dans la section 2, nous donnons les objectifs des deux mécanismes présentés, nommément la gestion de cache logiciel et la gestion de la réconciliation, et motivons la mise en place de ces services pour les applications réparties à base de composants. Ensuite, dans les sections 3 et 4, nous développons successivement les deux mécanismes en termes de méthodologie de conception puis de plateforme. Enfin, avant de présenter les travaux connexes et de conclure dans les sections 6 et 7, nous démontrons dans la section 5 l'intérêt de l'approche sur un scénario de gestion de crise développé dans le cadre du projet franco-finlandais AMPROS (http://www-inf.int-evry/AMPROS).

2. Motivations et objectifs

La construction d'applications réparties converge de plus en plus vers l'utilisation d'intergiciels orientés composants tels que les EJB [5], CCM [22] et .Net [19]. Par rapport au paradigme objet, le paradigme composant répond mieux au problème de la complexité de gestion des applications car il couvre toutes les étapes du cycle de vie des applications et offre une meilleure séparation entre les aspects fonctionnels et extrafonctionnels. Cette séparation est réalisée suivant le patron de conception composant/conteneur [26] : le composant encapsule les aspects fonctionnels et le conteneur gère les aspects extrafonctionnels.

[2] DOMINT est un projet Open Source sous licence GNU GPL dont les fichiers sources sont disponibles sur http://picolibre.int-evry.fr/.

Cependant, la gestion des aspects extrafonctionnels est soumise à de nombreuses limitations. Alors que les aspects fonctionnels peuvent être modifiés en toute liberté, les aspects extrafonctionnels sont généralement limités en ce qui concerne leur nombre, leur type et leurs interactions. De plus, ces composants ne peuvent pas s'adapter à leur contexte en modifiant les services extrafonctionnels au cours de l'exécution. Les aspects extrafonctionnels les plus utilisés sont la persistance, la gestion des transactions, la sécurité et la distribution. L'aspect de gestion de déconnexion, qui est l'objet de cet article, n'est que rarement considéré. Dans cet article, nous montrons que le principe de séparation des aspects fonctionnels et extrafonctionnels convient aussi à la gestion de déconnexion.

Dans une application répartie « classique » fonctionnant avec une bonne connectivité entre les différents composants, les composants proches de l'utilisateur peuvent s'exécuter sur des terminaux mobiles et n'être constitués que d'une interface graphique. Les autres composants de l'application (composants distants) restent sur des nœuds du réseau fixe. Pour y ajouter la continuité de service en présence des déconnexions, des mandataires des composants distants sont créés sur les terminaux mobiles. La disponibilité de ces mandataires offre une connexion logique entre le client et les serveurs en utilisant le concept d'opération déconnectée [14]. La continuité de service requiert la détection des déconnexions, la gestion d'un cache logiciel de composants déconnectés à créer avant les déconnexions, la journalisation et la réconciliation des opérations déconnectées après les reconnexions. Le premier mécanisme n'est pas décrit dans ce papier ; le lecteur peut se référer à [27].

Le service de gestion du cache du terminal mobile doit offrir deux stratégies importantes. Premièrement, la stratégie de déploiement détermine les composants déconnectés à créer dans le cache logiciel du terminal mobile, quand et pour quelle durée. Deuxièmement, la stratégie de remplacement décide quels composants déconnectés doivent être supprimés lorsqu'il n'existe plus assez d'espace mémoire dans le cache. En outre, l'application répartie est vue comme un assemblage de composants qui interagissent entre eux pour accomplir les fonctionnalités de l'application. Ces interactions sont vues comme des compositions logiques de composants qui définissent des services de l'application. Par conséquent, l'application est vue comme un ensemble de services auxquels les utilisateurs accèdent via la façade. De ce fait, nous définissons deux types d'interactions : intra-service (entre les composants d'un même service) et inter-services (entre les différents services de l'application). L'un des objectifs de la modélisation est donc de faire apparaître explicitement ces dépendances.

Le service de réconciliation doit prendre en compte les changements de connectivité du réseau. SOCT4 a été conçu et adapté pour intégrer trois modes d'interaction : synchrone, asynchrone et multi-synchrone [12]. Cependant, dans chacun des trois cas, le travail découplé est synonyme de passage en mode veille. Dans notre problématique, le travail isolé signifie la coupure physique du terminal mobile avec le reste du système. Par conséquent, nous fournissons dans cet article une stratégie de recouvrement permettant de récupérer les opérations ayant transité sur le réseau durant la période de travail hors connexion.

3. Modélisation de la gestion de déconnexion

Dans cette section, nous décrivons une méthodologie de conception d'applications réparties à base de composants telle que, respectivement, la gestion du cache logiciel (cf. section 3.1) et la gestion de la réconciliation (cf. section 3.2) s'adaptent efficacement aux variations de connectivité.

3.1. Gestion du cache logiciel

En ce qui concerne la gestion du cache logiciel, MADA (*Mobile Application Development Approach*) est une approche centrée sur l'architecture logicielle et pilotée par les cas d'utilisation (c.-à-d., partant des fonctionnalités fournies aux utilisateurs finaux). MADA suit l'approche MDA [22] de l'OMG comme l'indiquent la figure 1. Dans la figure 1-b, l'architecte construit une modélisation UML de l'architecture logicielle (dans la figure, le PIM, *Platform Independent Model*), tout en respectant le profil UML EDOC [22], *Enterprise Distributed Object Computing*, qui cible les applications réparties construites à base de composants. Comme indiqué, le PIM est indépendant de la plateforme d'implantation et d'exécution. Il est ensuite transformé en un modèle ayant pour cible le modèle de composant CORBA (CCM) et respectant le profil UML CCM [22]. Ne désirant pas fournir un atelier de génie logiciel, nous considérons que cette transformation existe. Par analogie, dans la figure 1-a, l'architecture logicielle de l'application est surchargée de spécifications dédiées à la gestion de déconnexion (dans la figure, le PIMDisc) et suit le profil EDOCDisc, lui-même étendu à partir de EDOC. Enfin, la projection dans le monde CCM donne le modèle PSMDisc respectant le profil UML4CCMDisc [17].

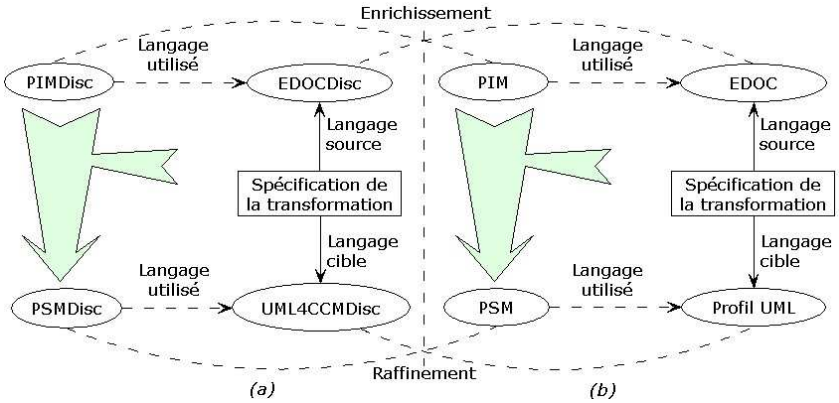

Figure 1 – Vision MDA de MADA.

MADA est basée sur le patron de conception « Façade » [8] et le modèle 4+1 vues de l'architecture [18]. Le patron de conception « Façade » permet de fournir un accès unifié à un groupe d'interfaces d'un sous-système. La façade réduit également le nombre de composants présentés au client. Ce patron de conception rend donc le sous-système plus facile à utiliser et à modifier. Le modèle 4+1 vues permet d'organiser la description d'une architecture logicielle en plusieurs vues concurrentes (logique, processus, physique, développement et cas d'utilisation). Chaque vue adresse séparément les soucis des divers intervenants (concepteurs, développeurs, architectes, utilisateurs…). Par ailleurs, le modèle 4+1 vues aide à la séparation des aspects fonctionnels et extrafonctionnels dès l'architecture.

Dans la suite de cette section, nous décrivons les méta-données du profil de l'application pour la déconnexion EDOCDisc (cf. section 3.1.1), puis la construction du graphe des dépendances entre composants et services avec la façade (cf. section 3.1.2), et enfin, la propagation des méta-données dans le graphe (cf. section 3.1.3).

3.1.1. Profil de l'application

Ce profil est basé sur trois méta-données : la déconnectabilité, la nécessité et la priorité. La *déconnectabilité* indique si un composant distant résidant sur un serveur fixe peut avoir un composant déconnecté sur le terminal mobile. Si c'est le cas, le composant distant est *déconnectable*. Un composant déconnecté réalise les mêmes fonctionnalités que le composant distant. L'architecte de l'application attribue la méta-donnée déconnectabilité aux composants car il possède la meilleure connaissance de la sémantique de l'application. En outre, la déconnectabilité implique des contraintes de conception que les développeurs doivent respecter. Par exemple, pour des raisons de sécurité, il est possible de déployer quelques composants sur des serveurs sécurisés et d'empêcher la duplication de ces composants sur des terminaux mobiles.

La *nécessité* permet de donner un « poids » aux composants déconnectés de l'application quant à leur présence dans le cache du terminal mobile. Un composant nécessaire est un composant dont la présence dans le terminal mobile est obligatoire durant les déconnexions. Un composant non nécessaire est un composant dont l'absence dans le terminal mobile ne peut pas empêcher le fonctionnement global de l'application en mode déconnecté. L'application doit donc être construite en conséquence. Contrairement à la méta-donnée *déconnectabilité*, la méta-donnée nécessité est dynamique. Le développeur fournit une première classification des composants déconnectés en « nécessaires développeur » et « non nécessaires ». Cette classification est utilisée pour construire un profil par défaut. En cours d'exécution, l'utilisateur de l'application peut personnaliser le profil en changeant la nécessité des composants non nécessaires pour qu'ils deviennent « nécessaires utilisateur ». Nous décrivons le changement dynamique de la nécessité dans la section 3.1.3.

Enfin, la méta-donnée *priorité* indique la priorité entre les composants non nécessaires et les composants nécessaires utilisateur. Cette méta-donnée est utilisée dans les stratégies de déploiement et de remplacement du cache, en particulier dans les situations critiques : par exemple, lors du choix des composants déconnectés à supprimer du cache lorsque ce dernier est saturé, ou lorsque la déconnexion a été prévue mais que le gestionnaire du cache n'a pas terminer de déployer les composants nécessaires utilisateurs.

Par analogie, nous appliquons ces méta-données au concept de service. Ainsi, nous définissons un service déconnectable comme un service de l'application qui peut être assuré durant les déconnexions. Un service déconnectable est la composition logique de plusieurs composants déconnectables. En outre, nous définissons un service nécessaire comme un service devant être assuré pendant les déconnexions. Un service nécessaire doit contenir au moins un composant nécessaire. Enfin, les services possèdent aussi une priorité.

3.1.2. Graphe de dépendances

Dans MADA, chaque service de l'application est représenté par un cas d'utilisation (diagramme UML des cas d'utilisation). Le comportement de l'application en présence des déconnexions est modélisé par des cas d'utilisation qui héritent des cas d'utilisation de base en utilisant la relation `extend`. Pour modéliser la méta-donnée nécessité, le profil UML4CCMDisc ajoute la valeur nommée `isNecessary` à la relation `extend` entre le cas d'utilisation de base et le cas d'utilisation spécialisé. Un cas d'utilisation peut inclure le comportement d'autres cas d'utilisation à l'aide de la relation `include`. Ainsi, l'utilisation d'un service en mode déconnecté peut exiger la présence d'autres services dans le cache logiciel du terminal mobile. La résolution de ce problème exige la détermination des dépendances entre les services. Nous modélisons ces dépendances dans un graphe orienté où les nœuds dénotent les services et les liens dénotent la dépendance entre les services. Dans ce graphe, les liens sont marqués par la nécessité. Dans ce cas, la nécessité d'un lien entre deux services (pré-service vers post-service) indique la nécessité du post-service pour le fonctionnement du pré-service en mode déconnecté.

Par ailleurs, la disponibilité d'un service en mode déconnecté implique la présence de certains composants dans le cache logiciel du terminal mobile. Ainsi, par analogie avec les services, les dépendances entre les composants sont aussi modélisés dans le graphe de dépendances où les nœuds dénotent les composants et les liens marqués par la nécessité dénotent les dépendances entre les composants. Le graphe de dépendances comporte quatre types d'interaction : entre la façade (la racine du graphe) et les services, entre les services (inter-services), entre les services est les composants (points d'entrée pour utiliser un service) et entre les composants (intra-service). Nous décrivons la propagation de la nécessité dans le graphe de dépendances dans la section suivante.

3.1.3. Propagation des méta-données dans le graphe de dépendances

La nécessité d'un service correspond à la nécessité entre la façade et le service : si le lien entre la façade et le service est nécessaire alors le service est nécessaire. Dans le cas où le lien entre deux services est nécessaire et le pré-service (service en amont) est nécessaire, la nécessité est propagée au post-service (service en aval). En ce qui concerne le passage de la nécessité du service aux composants contenus, si le service et le lien sont nécessaires alors le composant est nécessaire. Pour la propagation de la nécessité entre composants, si le lien entre deux composants est nécessaire et le pré-composant est nécessaire pour un service alors le post-composant devient aussi nécessaire pour ce même service. En outre, il est clair qu'un composant de l'application est nécessaire dès qu'il est nécessaire pour au moins un service nécessaire.

Dans le cas où l'utilisateur change la nécessité d'un lien entre la façade et un service (nécessaire utilisateur vers non nécessaire), et s'il n'existe pas d'autre service nécessaire qui utilise le premier service, ce premier devient non nécessaire. Considérons maintenant le cas où le service S_1, précédemment nécessaire, devient non nécessaire. Si le lien entre S_1 et un autre service S_2 est nécessaire et s'il n'existe pas d'autre service S_3 tel que S_3 et le lien entre S_3 et S_2 sont nécessaires, alors S_2 devient non nécessaire. Par ailleurs, dans le cas où il existe un lien entre un service S_1 et un composant C_1, si S_1 précédemment nécessaire devient non nécessaire, s'il n'existe pas d'autre service S_2 tel que S_2 et le lien entre S_2 et C_1 sont nécessaires et s'il n'existe pas d'autre composant C_2 tel que C_2 et le lien entre C_2 et C_1 sont nécessaires, alors C_1 devient non nécessaire. Enfin, dans le cas où il existe un lien entre un composant C_1 et un composant C_2, si C_1 précédemment nécessaire devient non nécessaire, s'il n'existe pas de service S_1 tel que S_1 et le lien entre S_1 et C_2 sont nécessaires et s'il n'existe pas d'autre composant C_3 tel que C_3 et le lien entre C_3 et C_2 sont nécessaires, alors C_2 devient non nécessaire.

3.2. Gestion de la réconciliation

Dans un contexte mobile, la continuité de fonctionnement de l'application indépendamment de la qualité de la connexion est fondamentale. Dans ce cas, la technique de réplication optimiste permet à plusieurs utilisateurs de travailler sur une copie de l'application déployée sur leur terminal mobile sans qu'ils ne soient gênés par d'éventuels verrous comme dans le cas de la réplication pessimiste. Cette approche offre

des avantages en terme de disponibilité de la donnée partagée et de performance. Cependant, ce mode de travail déconnecté autorise l'évolution en parallèle des copies lors des périodes de travail hors connexion. La conséquence est l'apparition de divergences entre les copies nécessitant la mise en place d'une stratégie de réconciliation afin de synchroniser les entités cachées et distantes à la reconnexion.

3.2.1. Présentation générale de SOCT4

Notre approche de réconciliation pour la gestion de divergence est fondée sur l'algorithme SOCT4 (*Sérialisation des Opérations Concurrentes par Transposition*) [30], conçu pour fonctionner dans le cadre du travail collaboratif. Il est basé sur la technique de réplication optimiste qui est particulièrement bien adaptée pour répondre efficacement au problème de la perte de la connectivité en assurant une continuité de service et fait appel à la technique des transformées opérationnelles. Ces transformées exploitent les propriétés sémantiques des opérations pour les sérialiser en vue d'aboutir à la cohérence entre les différentes répliques de l'application partagée. Pour garantir des histoires identiques sur tous les sites au terme de l'étape de réconciliation, les trois conditions suivantes doivent être satisfaites : la préservation de la causalité, de l'intention de l'utilisateur et la convergence. Celles-ci sont résolues respectivement grâce à la procédure de réception causale qui repose sur un ordre global (cf. section 4.2), à la fonction de transposition en avant et au respect de la condition C1 [3, 30]. Dans cette condition, l'exécution d'une opération (op_1) suivie d'une autre opération (op_2) sur un site doit produire le même résultat que l'exécution de op_2 suivie de op_1 sur un autre site.

Une des problématiques de cet algorithme est que les fonctions de transformation écrites sont fortement corrélées avec la sémantique des opérations de l'application cible. Cette contrainte a été relâchée grâce à la modélisation XML [12]. Le compromis à réaliser est l'écriture, par le concepteur de l'application, des correspondances entre les opérations XML et les opérations pertinentes exécutées sur le composant partagé. Cette généricité rend notre solution adaptable à n'importe quelle application dont la donnée partagée peut être facilement représentée en XML. Nos précédents travaux ont abouti à la modélisation XML d'une application de messagerie électronique présentée dans [3].

3.2.2. Adaptation de SOCT4 à la mobilité

Une extension de l'algorithme SOCT4 intégrant l'aspect mobilité et fournissant les procédures à suivre dans les cas de déconnexions volontaires, involontaires et de reconnexion est fournie dans [29]. Le moyen proposé pour traiter les déconnexions et reconnexions repose sur la connaissance et l'interrogation d'un site fixe connu qui jouera le rôle de mandataire. Cependant, cette solution est très contraignante du fait que le site fixe est supposé être disponible au moment de la requête. C'est pourquoi, nous proposons dans cette section une adaptation de l'algorithme SOCT4. Cette solution, efficace en présence d'équipements mobiles, ne fait aucune hypothèse sur la topologie du réseau et ne nécessite pas la présence d'un site fixe.

C'est à la reconnexion, que la stratégie de réconciliation est initiée. Elle se décline en deux opérations. Dans un premier temps, l'opération *update* effectue l'intégration des opérations distantes permettant la mise à jour du composant local. Dans un deuxième temps, l'opération *commit* autorise la diffusion des opérations locales aux pairs. Notre approche adresse le recouvrement des opérations distantes, échangées durant la période de travail déconnecté, via l'opération *update* (cf. figure 2).

Soit *G* le groupe de travail dans lequel l'application cible a été répliquée et modifiée et *Ld* le ticket de la dernière opération dans l'histoire du site A. L'algorithme de recouvrement s'articule en quatre étapes. Dans la première étape, le site A se reconnectant, déclenche la procédure de recouvrement des opérations distantes en diffusant à *G* son *Ld* grâce à la procédure *RequestHistory*. Dans la deuxième étape, les sites B, C et D recevant la requête comparent leur propre *Ld* appelé *LdLocal* à celui reçu. Si *Ld* et *LdLocal* sont équivalents, alors il n'y a rien à faire. Sinon, si *Ld* est inférieur à *LdLocal* alors le site interrogé déclenche à son tour la procédure de recouvrement. Enfin, si *Ld* est supérieur à *LdLocal* alors le site renvoie la valeur de son *LdLocal* à A grâce à la procédure *SendHistory*. Dans la troisième étape, le site initiateur range chaque réponse dans une table et contacte le premier site qui possède le plus grand *Ld* en utilisant la procédure *RequestDiff* afin de récupérer toutes les opérations distantes échangées durant son travail isolé. Enfin, dans la dernière étape, le site interrogé, dont la réplique est la plus « fraîche », retourne au site A la partie de l'histoire manquante comprise entre *LdStart* et *LdEnd*.

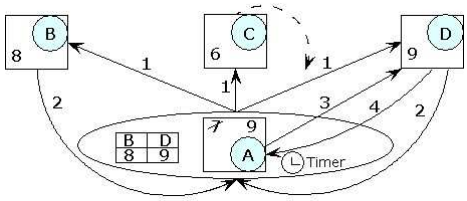

Figure 2 – Procédure de recouvrement.

4. DOMINT

L'objectif de DOMINT est d'offrir la continuité de service en présence de déconnexions pour applications à base de composants CORBA. Chaque composant de l'application est contrôlé par un conteneur dédié à la gestion des déconnexions [16]. Nous proposons quatre services extrafonctionnels utilisés de manière transparente pendant l'exécution : le gestionnaire du cache, le détecteur de connectivité, le détecteur de déconnection et le gestionnaire de réconciliation. Nous avons utilisé le modèle de composant Fractal [23] pour la modélisation et l'implantation de ces services. Dans cet article, nous nous limitons à la description des gestionnaires du cache (cf. section 4.1) et de réconciliation (cf. section 4.2).

4.1. Architecture du gestionnaire de cache

Le gestionnaire du cache centralise la gestion de tous les composants déconnectés dans le cache du terminal mobile. Nous avons choisi de partager le cache pour toutes les applications s'exécutant sur le terminal mobile afin d'éviter d'avoir plusieurs instances d'un même composant déconnecté. La figure 3-a décrit l'architecture du *CacheManager*. C'est un composite constitué de quatre composants Fractal. Le composant *DisComponentFactory* représente le point d'entrée du *CacheManager*. Il coordonne le déploiement des services et des composants dans le cache. Nous définissons trois stratégies de déploiement : au lancement de l'application, au cours de l'exécution de l'application et à l'invocation [15]. Le composant *DisComponentCreator* contrôle la création des composants déconnectés. Il utilise le mécanisme de déploiement d'OpenCCM [24]. Le composant *DiscEntryFactory* contrôle les entrées du cache. Chaque entrée du cache encapsule la référence CORBA du composant distant, la référence CORBA du composant déconnecté et les méta-données.

Nous avons utilisé le composant *PerseusCacheManager* de *Perseus* [23]. La structure du composant *PerseusCacheManager* est schématisée sur la figure 3-b. Le composant *DiscReplacementManager* implante le composant abstrait *ReplacementManager* de *Perseus* afin d'intégrer nos stratégies de remplacement. Nous avons proposé dans [15] deux stratégies de remplacement : le remplacement à la demande de l'utilisateur et le remplacement périodique.

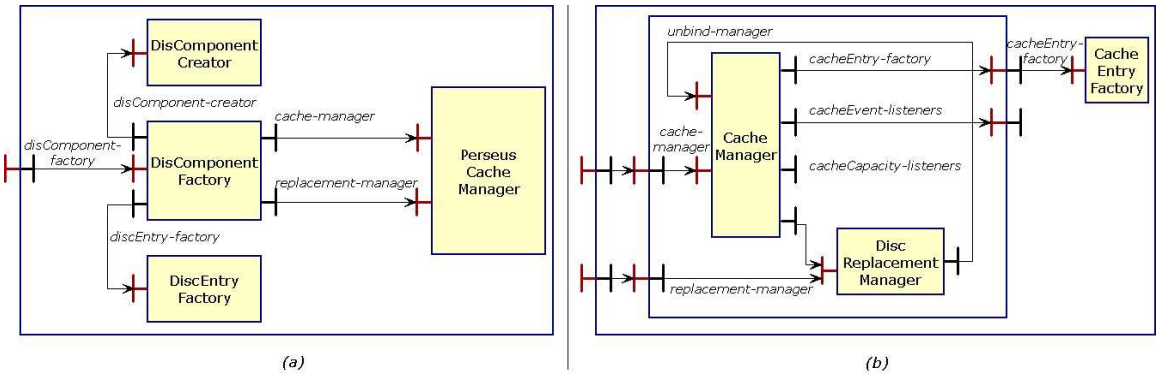

Figure 3 – *(a)* CacheManager *(b)* Perseus.

4.2. Architecture du gestionnaire de la réconciliation

Le service de réconciliation gère toutes les opérations invoquées sur la copie afin de les traiter conformément à l'algorithme SOCT4. Cet algorithme distingue deux types d'opérations : les opérations locales que l'utilisateur propriétaire de la copie a généré ainsi que les opérations distantes exécutées par les autres utilisateurs et à répercuter sur la copie. Les opérations locales sont interceptées par un contrôleur

dédié via la facette locale associée à la réplique. Les opérations distantes sont reçues par le composant BroadcastManager (cf. figure 4).

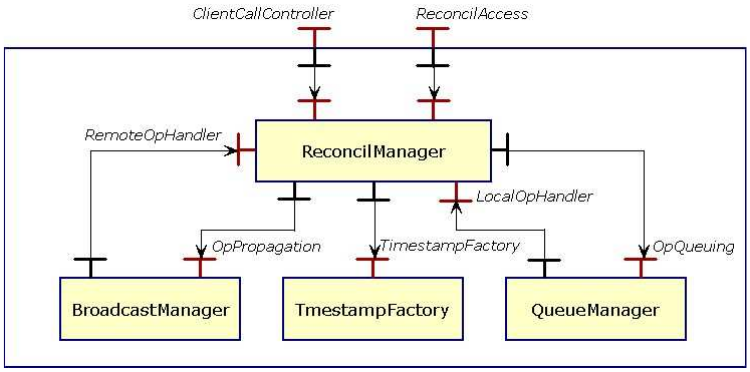

Figure 4 – Architecture du ReconciliationManager.

Le composant *ReconcilManager* réalise l'algorithme de réconciliation et coordonne son processus en s'appuyant sur les autres composants du service de réconciliation. Il active le processus de réconciliation et prépare le travail local durant la déconnexion. L'interface *RemoteOpHandler* permet au *ReconcilManager* d'être notifié de l'arrivée d'une nouvelle opération locale ou distante dans leurs files respectives. Il pourra ainsi procéder à la réconciliation des opérations au fur et à mesure de leur arrivée quand l'utilisateur est connecté. En outre, via l'interface *LocalOpHandler*, il récupère les opérations générées durant la déconnexion. Le composant *QueueManager* gère les files d'attente des opérations journalisées. Les opérations sont ordonnées selon une estampille. Le composant *BroadcastMger* utilise Javagroups [1] pour diffuser les opérations selon l'ordre total défini par le séquenceur *TimestampFactory*, reposant sur un séquenceur réparti tolérant aux fautes [2]. Toutes les opérations de tous les utilisateurs peuvent ainsi être ordonnées de façon unique.

5. Scénario

Nous présentons dans cette section un scénario sur la gestion de catastrophes et montrons comment les services présentés précédemment peuvent être mis en œuvre pour faciliter le travail des équipes de secours, personnels de santé et forces de sécurité, relevant de deux autorités différentes. Ce scénario est inspiré des méthodes de travail des centres des urgences finlandais et français dans le cadre du projet AMPROS.

Les acteurs et leur équipement Nous considérons que les acteurs en présence sont munis de terminaux sans fil (assistants personnels numériques ou ordinateurs portables) pouvant communiquer en mode WIFI et/ou GSM/GPRS. Les médecins secouristes, sous le commandement du DSM (Directeur des Secours Médicaux) sont les premiers à intervenir auprès des victimes. Ils sont chargés d'équiper les victimes de couvertures appareillées et munies de capteurs permettant un suivi en temps réel des fonctions vitales (rythme cardiaque, température, tension artérielle...). Leurs autres attributions sont de créer une fiche médicale de l'avant (du terrain) pour chaque victime, contrôler l'évolution des informations fournies par les capteurs, consulter le dossier médical de la victime s'il est disponible, et faire un premier bilan de l'état de santé des victimes. Ce bilan sert à classer les victimes en fonction de la gravité de leurs blessures. D'un point de vue médical, le DSM a pour mission d'avaliser le classement des victimes dans les catégories de blessés, affecter le personnel de santé aux différentes zones et suivre leurs interventions, faire établir la liste des capacités d'accueil hospitalier, gérer l'évacuation vers les établissements de santé, contrôler la situation, et rendre compte de l'évolution de la situation du point de vue médical au COD (Centre Opérationnel Départemental). D'un point de vue logistique, le COS (Commandant des Opérations de Secours) est chargé de prendre les décisions concernant les équipements (ambulances, postes de secours...), d'évaluer la situation avec le DSM et le premier médecin sur les lieux et d'en rendre compte, et de demander la déclenchement d'un plan rouge si nécessaire. Gérant globalement les secours sur le terrain, le COD est une cellule de gestion des opérations regroupant un représentant de l'autorité préfectorale, un représentant des autorités sanitaires et sociales, un représentant du procureur et le directeur du SAMU. Cette cellule peut être présente sur le site dans un poste de commandement local. Le responsable du COD dispose de différentes vues de synthèse du déroulement des secours : nombre de victimes et gravité de leurs blessures, nombre et identification des victimes décédées, répartition des moyens, disponibilité des ressources dans les centres de santé...

Applications utilisées par les acteurs Les victimes sont équipées de capteurs divers et variés tous reliés à une unité centrale accessible par un réseau sans fil personnel (par exemple, Bluetooth). Les informations venant des capteurs plus les informations ajoutées par les divers intervenants constituent la fiche médicale de l'avant. Les assistants personnels numériques ou terminaux mobiles ne sont pas configurés pour tel ou tel intervenant, si ce n'est qu'ils contiennent le canevas logiciel DOMINT. En d'autres termes, les composants des applications sont chargés à la demande par les gestionnaires de caches logiciels. Les terminaux sont donc personnalisés au fur et à mesure de leurs utilisations et sont échangeables, par exemple en cas de défaillance matérielle. Cela n'empêche pas de classer les applications par catégorie d'intervenants.

Les médecins sauveteurs ont besoin de manipuler les fiches médicales de l'avant et d'interpréter les données fournies par les capteurs. Ils doivent pouvoir demander conseil au spécialiste de médecine d'urgence qu'est le DSM, voire aux spécialistes des services hospitaliers vers lesquels les victimes sont évacuées. Cette demande d'aide est effectuée via une application de conférence par messages (en anglais *chat*), voire par vidéo. Lors des déconnexions, les médecins secouristes peuvent utiliser un système expert.

Le DSM a besoin de visualiser géographiquement les médecins secouristes afin de les orienter. Il obtient via les médecins secouristes les localisations et les états des victimes. Cela lui permet ainsi de coordonner les actions sur le terrain avec le médecin responsable de la zone de tri et d'évacuation des victimes. Le COS possède une application similaire pour la coordination des actions logistiques sur le terrain. Ces deux responsables se synchronisent aussi régulièrement en échangeant les états résumés sur les victimes et sur le dispositif logistique. Cette coordination est subordonnée aux décisions du COD, qui est l'entité responsable officielle.

Scénario de suivi médical des victimes avec déconnexion Pour des raisons de performance et pour permettre la continuité de service en dépit de la fluctuation de la connectivité réseau, plusieurs copies de la fiche médicale sont présentes dans le système au risque de l'apparition d'une certaine divergence. Le service de réconciliation permettra donc de maintenir la cohérence entre les différentes copies de la fiche médicale.

Une fiche médicale est créée pour chaque nouvelle victime secourue, indiquant notamment un premier diagnostic avec le type et l'importance des blessures constatées, ainsi que les soins à pratiquer. La fiche est initialement créée au niveau de la couverture équipant la victime. Cette fiche reçoit un numéro d'identification unique immédiatement si la connectivité réseau est suffisante afin de demander ce numéro à un service d'estampillage ; dans le cas contraire, la numérotation sera simplement différée jusqu'à ce que la connectivité soit meilleure.

Dès que la fiche est créée, elle peut être téléchargée automatiquement par les secouristes à portée de la victime. Ceux-ci peuvent ainsi la consulter et la mettre à jour si nécessaire. Si la connectivité le permet, ceci entraînera une modification également de la fiche présente au niveau de la couverture des patients. Des mises à jour sont également effectuées directement par les capteurs indiquant les fonctions vitales des victimes.

Nous nous intéressons maintenant au cas d'une déconnexion survenant au niveau de l'un des sauveteurs après avoir chargé une fiche médicale et avant qu'il n'indique son diagnostic. Nous distinguons quatre phases dans la gestion des déconnexions : une phase de préparation en vue de déconnexions possibles, une phase de travail local ou isolé, une phase d'intégration des opérations distantes, et une phase de collaboration avec les autres acteurs du système.

Phase 0 — Préparation de la déconnexion Dès que le médecin secouriste prend possession de son terminal mobile, il s'identifie. Selon son profil, un ensemble de services est déployé sur le terminal mobile. Les services sont démarrés et les connexions avec les composants serveurs distants ouvertes. Il peut ensuite modifier la configuration de départ pour demander quelques services supplémentaires comme le système expert. Ces derniers services sont démarrés si le gestionnaire de cache l'autorise. Pendant son activité, le médecin utilise des composants initialement définis non nécessaires comme un visualisateur graphique (plutôt que textuel) de l'historique des informations d'un capteur pour une victime. Ces composants sont démarrés à la première utilisation. En contre partie, le gestionnaire du cache logiciel arrête si besoin les services non nécessaires non prioritaires et non utilisés. Il est à noter que les informations évoluent dans le

temps et peuvent être partagées avec d'autres intervenants. Tous les composants déconnectés sont régulièrement synchronisés avec les composants distants.

Phase 1 — Travail local Lorsqu'une déconnexion est détectée, si la déconnexion est volontaire, le service de gestion de cache met à jour une dernière fois les composants déconnectés. Si la déconnexion est subite et involontaire, le médecin continue à travailler avec les dernières données obtenues avant la déconnexion. De même, le composant *ReconcilManager* du service de réconciliation prépare plusieurs files pour stocker les opérations effectuées en mode déconnecté. Lorsque l'opération de mise à jour de la fiche médicale est effectuée, celle-ci est interceptée par le service de réconciliation qui la transmet au *QueueManager* pour qu'il la place dans la file des opérations locales.

Phase 2 — Intégration Au moment de la reconnexion, le *ReconcilManager* commence par récupérer les opérations distantes ainsi qu'il a été présenté à la section 3.2.2 et les places dans la file de réception des opérations. Ces opérations sont ensuite transposées selon la fonction de transformation avant de SOCT4 et sont exécutées sur le composant local. Puis, les opérations effectuées pendant la phase 1 sont traitées. Le composant *TimestampFactory* demande une estampille au séquenceur global pour chacune des opérations locales puis les diffusent au reste du système via le *BroadcastManager*.

Phase 3 — Collaboration Après une reconnexion, la collaboration entre les différents acteurs du système est de nouveau possible. À chaque fois qu'une opération est demandée par l'application, elle est immédiatement exécutée puis placée dans la file des opérations locales par le *QueueManager*. Une estampille est ensuite demandée au séquenceur par le *TimestampFactory*. Par ailleurs, le *ReconcilManager* peut être informé par le *BroadcastManager* que des opérations distantes ont été reçues. Celles-ci sont placées dans la file de réception, éventuellement transformées puis exécutées. Si toutes les opérations précédentes ont été reçues, l'opération locale est alors diffusée par le *BroadcastManager*.

6. Travaux connexes

La problématique traitée dans cet article aborde les domaines de recherche autour de la mobilité des applications, en particulier, la gestion des déconnexions. Nous étudions les différentes propositions par orientation : fichier, base de données, objet et composant.

Coda [25] est un système de gestion de fichiers qui définit la notion de données implicites et de données explicites. Les données implicites représentent l'historique d'utilisation du client. Les données explicites prennent la forme d'une base de données appelée *HDB* (*Hoard Data Base*) construite par le client de l'application. Le gestionnaire du cache de Coda appelé *Venus* fonctionne sous trois modes : accumulation, émulation et réintégration. En mode accumulation (mode connecté), Coda anticipe les déconnexions en copiant localement les fichiers du *HDB* selon une priorité accordée par l'utilisateur. Les autres fichiers sont chargés sur un défaut d'accès. Cependant, si l'utilisateur fait un mauvais choix suite à une mauvaise compréhension de l'application, cette dernière peut ne pas fonctionner en mode déconnecté. Dans notre approche, nous avons ajouté la construction du profil de l'application répartie effectuée par l'architecte. Coda utilise aussi une stratégie optimiste de contrôle des répliques. Il assure la détection des incohérences des mises à jour et fournit des mécanismes d'aide aux utilisateurs pour éliminer ces incohérences. Les opérations sont limitées aux opérations sur les fichiers.

En utilisant Coda comme système de fichiers, Seer [9] propose une stratégie prédictive de déploiement. Il se base sur le concept de *distance sémantique* pour définir la dépendance entre les fichiers. Seer regroupe en *projets* les fichiers ayant entre eux une faible distance sémantique. Il observe le comportement de l'utilisateur, calcule les distances sémantiques, génère des projets et emploie ces projets pour préparer le cache logiciel en vue des déconnexions. Bien que le projet définisse une dépendance logique entre les fichiers, Seer ne définit pas de dépendances entre projets. Odyssey [21] ajoute pour chaque objet distant de l'application deux méta-données : « fidélité » et « fenêtre de tolérance ». Ces méta-données sont bien adaptées pour la gestion de la faible connectivité, mais pas pour les déconnexions.

Le projet IceCube [13] utilise la réconciliation basée sur les spécificités sémantiques de l'application et l'intention de l'utilisateur. Cette technique exploite des journaux pour trouver un ordre dans les opérations enregistrées. Durant la phase de réconciliation, le système les fusionne de manière à minimiser les conflits et amener les copies vers un état cohérent. Le processus de réconciliation se déroule en trois étapes. L'ordonnancement (ou la fusion des journaux) considère toutes les combinaisons possibles respectant les contraintes statiques entre les actions. La simulation consiste à exécuter les séquences d'actions déterminées lors de la première étape sur une copie de l'objet partagé. La sélection choisit parmi les

résultats de la simulation ceux qui satisfont l'application. Néanmoins, ce système soulève le problème de l'explosion combinatoire induit par l'exploitation de tous les ordonnancements possibles. En outre, il impose de restreindre le champ de recherche par des contraintes statiques.

Bayou [29] propose un mécanisme de réplication faiblement cohérent pour un groupe d'utilisateur. Il utilise toute la base de données comme granularité de réplication. Ainsi, chaque terminal mobile doit posséder une copie de la base de données (les tables et les enregistrements) pour le fonctionnement en présence des déconnexions. La détection et la résolution des incohérences sont assurées par un mécanisme d'anti-entropie et des procédures de résolution fournies par l'application. Cependant, l'ordonnancement des opérations est arbitraire et n'exploite pas la sémantique associée aux opérations. Elle peut conduire à un accroissement du journal et impliquer de faire et défaire des opérations intégrées, ce qui est évité par l'utilisation de SOCT4. [7] propose un mécanisme de gestion du cache logiciel dans le contexte des bases de données orientée objets. Le déploiement et le remplacement dans le cache sont basés sur la demande de l'utilisateur. Cependant, ce mécanisme ne prévoie pas les déconnexions et ne prend pas en compte la sémantique de l'application.

Rover [11] définit les notions d'objet dynamique relogeable et d'appel de procédure à distance non bloquant. Rover traite tous les objets de l'application de la même façon et ne tient pas compte de la sémantique de l'application. Rover gère la cohérence en utilisant des mécanismes similaires à ceux de Bayou. CASCADE [4] définit un service générique et hiérarchique de gestion de serveurs mandataires pour des objets CORBA. Il définit un service de désignation pour les serveurs mandataires et les clients utilisent ce service pour trouver les serveurs mandataires. Lors des déconnexions, le client ne peut pas continuer à travailler si sa machine n'est pas un serveur de mandataires.

Nos travaux sur la réconciliation et ceux de [20] ont l'approche transformationnelle en commun. En effet, tous deux s'appuient sur les trois propriétés fondamentales que sont la convergence, la causalité et la préservation de l'intention de l'utilisateur. Cependant, notre objectif principal n'est pas de développer des fonctions de transformations et de les prouver formellement. Notre approche vise à étendre la portée de l'utilisation de l'outil de synchronisation sûr et générique SOCT4 afin de l'employer pour réconcilier des copies d'applications nomades réparties à base de composants.

7. Conclusion et perspectives

Nous avons proposé dans ce papier une méthodologie et une conception pour la gestion de cache logiciel et la gestion de réconciliation dans le cadre de déconnexions en environnement mobile. Ces travaux ont conduit à la définition d'une infrastructure à base de composants logiciels pour un service de mobilité.

Pour la gestion du cache de composants logiciels, nous avons proposé la méthodologie de conception MADA mettant en œuvre un ensemble de méta-données pour la construction d'un profil de l'application répartie. Dans le cadre de la plateforme DOMINT et en utilisant l'approche MADA, nous avons modélisé et réalisé un service de gestion du cache logiciel pour composants CORBA.

Les atouts principaux du service de gestion de réconciliation sont d'une part son indépendance vis-à-vis de l'application puisque notre proposition repose sur le concept de conteneur, et d'autre part, son adaptabilité aux spécificités de l'application de par le confinement de l'algorithme dans un composant unique. Notre contribution est complétée par la proposition d'une stratégie de recouvrement, à la reconnexion, afin que le site se reconnectant se synchronise avec ses pairs.

La plateforme que nous développons sera mise en œuvre sur le scénario de gestion de crise que nous avons présenté dans le cadre du projet RNTL ProAct franco-finlandais AMPROS.

Références

[1] JGroups Home page. http://www.jgroups.org/javagroupsnew/docs/index.html.
[2] R. Baldoni, C. Marchetti, and S. T. Pergiovanni. A Fault-Tolerant Sequencer for Timed Asynchronous Systems. In *Euro-Par 2002, LNCS 2400*, Berlin Heidelberg, Germany, 2002.
[3] L. Chateigner, S. Chabridon, and G. Bernard. Intergiciel pour l'informatique nomade : réplication optimiste et réconciliation. In *Manifestation des jeunes chercheurs STIC, MAJECSTIC*, Marseille, France, October 2003.
[4] G. Chockler, D. Dolev, R. Friedman, and R. Vitenberg. Implementing a caching service for distributed CORBA objects. In *Proc. 2nd IFIP/ACM Middleware*, pages 1–23, New York, USA, April 2000.

[5] L. DeMichiel. *Enterprise JavaBeans Specifications, version 2.1, proposed final draft*. Sun Microsystems, http://java.sun.com/products/ejb/docs.html, August. 2002.
[6] G. Forman and J. Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(4) :38–47, April 1994.
[7] M. Franklin. Transactional Client-Server Cache Consistency: Alternatives and Performance. *ACM Transactions on Database Systems*, 22(3) :315–363, September 1997.
[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.
[9] H. Geoffrey and J. Gerald. Automated Hoarding for Mobile Computers. In *Proc. 16th ACM SOSP*, pages 264–275, Saint-Malo, France, October 1997.
[10] J. Jing, A. Helal, and A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2), June 1999.
[11] A. Joseph, J. Tauber, and M. Kaashoek. Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers, special issue on mobile computing*, 46(3) :337–352, 1997.
[12] S. Jourdain. SAMS : Environnement coopératif Synchrone, Asynchrone, Multi-Synchrone pour les équipes virtuelles. Rapport de DEA, Université UHP Nancy-1, France, July 2002.
[13] A. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The IceCube approach to the reconciliation of divergent replicas. In *Proceedings of the 20th ACM PODC*, Newport, Rhode Island, USA, August 2001.
[14] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In *Proc. 13th ACM SOSP*, pages 213–225, Pacific Grove, USA, May 1991.
[15] N. Kouici, D. Conan, and G. Bernard. Caching Components for Disconnection Management in Mobile Envrionments. In Z. Tari *et al*, editor, *Proc. 6thDOA*, volume 3291 of *Lecture Notes in Computer Science*, pages 1322–1379, Agia Napa, Cyprus, October 2004. LNCS3291.
[16] N. Kouici, D. Conan, and G. Bernard. Intégration d'un service de gestion de déconnexions dans les conteneurs des composants. In *Journées Composants*, Lille, France, March 2004.
[17] N. Kouici, N. Sabri, D. Conan, and G. Bernard. MADA, une approche pour le développement d'applications mobiles. In *Proc. UbiMob*, Nice, France, June 2004. ACM International Conference Proceedings Series.
[18] P. Kruchten. Architectural Blueprints : The 4+1 View Model of Software Architecture. *IEEE Software*, 12(6) :42–50, November 1995.
[19] Microsoft. Microsoft Developer Network. http ://www.msdn.microsoft.com.
[20] P. Molli, G. Oster, H. Skaf-Molli, and A. Imine. Using the Transformational Approach to Build a Safe and Generic Data Synchronizer. In *Group'03*, Sanibel Island, Florida, USA, November 2003.
[21] B. Noble and M. Satyanarayanan. Experience with Adaptive Mobile Applications in Odyssey. *Mobile Networks and Applications*, 4(4) :245–254, 1999.
[22] Object Management Group. OMG home page. http ://www.omg.org, 2004.
[23] ObjectWeb Open Source Software Community. ObjectWeb home page. http ://www.objectweb.org, 2004.
[24] E. Pitoura and B. Bhargava. Building Information Systems for Mobile Environments. In *Third ACM International Conference on Information and Knowledge Management*, pages 371–378, Gaithersburg, Maryland, USA, November 1994.
[25] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Proc. 15thACM PODC*, pages 1–7, Philadelphia, Pennsylvania, USA, May 1996.
[26] C. Szyperski, D. Gruntz, and S. Murer. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 2002.
[27] L. Temal and D. Conan. Détections de défaillances, de connectivité et de déconnexions. In *Proc. UbiMob*, Nice, France, June 2004. ACM International Conference Proceedings Series.
[28] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing Update Conflicts in Bayou: A Weakly Connected Replicated Storage System. *Proc. 15th ACM SOSP*, Colorado, USA, December 1995.
[29] N. Vidot. Convergence des Copies dans les Environnements Collaboratifs Répartis. Thèse de doctorat, Université Montpellier-2, Montpellier, France, September 2002.
[30] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proc. ACM CSCW*, Philadelphia, Pennsylvania, USA, December 2000.