ADAPTATION DES APPLICATIONS REPARTIES A BASE DE COMPOSANTS AUX TERMINAUX MOBILES EN ENVIRONNEMENT SANS FIL

Nabil Kouici, Denis Conan et Guy Bernard

GET / INT, CNRS Samovar
9 rue Charles Fourier, 91011 Évry, France
{Nabil.Kouici, Denis.Conan, Guy.Bernard}@int-evry.fr
Télé : 33 1 60 76 44 15, 45 34, 45 67
Fax : 33 1 60 76 47 80

Résumé : les terminaux mobiles tels que les assistants personnels numériques ou les téléphones portables sont de plus en plus utilisés. Cependant, l'accès aux applications réparties à partir de ces terminaux soulève le problème de la disponibilité de ces services en présence des déconnexions. Le système et les applications distribuées fonctionnant sur ces terminaux doivent s'adapter aux changements de l'environnement mobile. Le travail présenté dans cet article est la continuation de Domint, une plate-forme pour la gestion des déconnexions pour les applications à base d'objets CORBA. Cet article propose une approche pour la gestion des déconnexions pour des applications à base de composants CCM (*CORBA Component Model*). Nous proposons un patron de conception pour le partitionnement des entités de l'application. Ce partitionnement est basé sur deux méta-données : « déconnectabilité » et « nécessité ». Ensuite, nous présentons D❋MINT, une architecture qui permet l'adaptation des applications réparties à base de composants CCM aux déconnexions dans les environnements mobiles.

Summary : The popularity of mobile terminals such as personal digital assistants and mobile phones raises the problem of the service availability in the presence of disconnections. The system and the distributed applications running on these terminals must adapt to mobile environment changes. The work presented in this article is the continuation of Domint, a platform to cope with disconnections in mobile environments for CORBA-objects-based applications. In this article, we propose a promising approach for the disconnections management of CCM-based applications (*CORBA Component Model*). We propose a design pattern for the partitioning of application entities. This partitioning is based on two metadata: "disconnectability" and "necessity". Finally, we present D❋MINT, a platform to cope with the disconnections in mobile environments for CCM-based applications.

Mots-clés : Mobilité, intergiciel, déconnexion, composant, méta-donnée.

Adaptation des applications réparties à base de composants aux terminaux mobiles en environnement sans fil

1 - INTRODUCTION

Ces dernières années ont été marquées par une forte évolution des équipements utilisés dans les environnements répartis. Nous sommes successivement passés de réseaux locaux à des réseaux à grande échelle (Internet), puis à des réseaux sans fil interconnectant des machines mobiles comme des téléphones portables ou des assistants personnels numériques (PDA). Cette évolution a abouti à la définition d'une nouvelle technologie qui se base sur l'infrastructure des réseaux mobiles. Cette technologie est l'informatique mobile dans laquelle l'utilisateur peut continuer d'accéder à l'information fournie par une infrastructure distribuée, sans tenir compte de son emplacement.

La construction d'applications réparties converge de plus en plus vers l'utilisation des intergiciels (*middleware*) à composants, en particulier les modèles de composants CORBA [Object Management Group, 2002] de l'OMG, EJB [DeMichiel, 2002] de SUN et .NET [Microsoft, 2002] de Microsoft. Le modèle de composants offre une séparation entre les fonctionnalités métier du composant et les fonctionnalités système (sécurité, transaction, cycle de vie,…) de l'intergiciel. Cette séparation est réalisée suivant le paradigme composant/conteneur. En effet, le composant encapsule le code métier et le conteneur gère les aspects extra-fonctionnels (fonctionnalité système) [Szyperski *et al.*, 2002].

Le travail présenté dans cet article est la continuation de Domint [Conan *et al.*, 2002a, Conan *et al*., 2002b], une plate-forme à base d'objets CORBA qui offre aux applications mobiles la continuité de services en mode déconnecté dans des environnements mobiles. L'idée principale de Domint est de créer sur le terminal mobile des entités mandataires appelées « objets déconnectés ». Dans Domint, la gestion de la connectivité utilise un mécanisme d'hystérésis qui donne les différents niveaux de connectivité (connecté, partiellement connecté et déconnecté).

La contribution principale des travaux présentés dans cet article est la proposition d'une architecture à base de composants nommée D✳MINT pour la gestion de la déconnexion. Dans cette architecture, nous proposons un patron de conception pour la classification des entités d'une application répartie, pour distinguer les entités qui peuvent avoir une entité déconnectée dans le terminal mobile et sélectionner les entités nécessaires dans le terminal mobile pour que l'application continue à fonctionner en mode déconnecté. Nous proposons aussi un mécanisme pour la création et la gestion des entités déconnectées et un mécanisme pour opérer le transfert d'états entre les entités déconnectées et les entités distantes.

La suite de l'article est divisée comme suit. La section 2 identifie nos motivations et objectifs. Ensuite, dans la section 3, nous présentons les métadonnées que nous utilisons dans la section 4 dans le contexte des intergiciels orientés composants. La section 5 décrit brièvement l'architecture D✳MINT. La section 6 établit le lien avec les travaux existants. Finalement, la section 7 conclut l'article et donne quelques perspectives.

2 - MOTIVATIONS ET OBJECTIFS

Les terminaux mobiles devenant extrêmement courants, leur utilisation ne cesse de se généraliser dans tous les domaines. Leur utilisation doit donc devenir aussi naturelle que possible. En particulier, la variation de la connectivité ne doit pas être vue comme une faute dans les environnements mobiles. Pour des applications distribuées, la variation de connectivité peut se traduire par 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, l'énergie et la minimisation des désagréments induits par les déconnexions inopinées. Les secondes sont le résultat de coupures intempestives des déconnexions physiques du réseau, par exemple, lors de passage de l'utilisateur dans une zone d'ombre radio.

Les environnements d'exécution d'applications réparties qui se basent sur les intergiciels actuels reposent sur des hypothèses (utilisateurs fixes, terminaux puissants, connexions réseau de bonne qualité et peu coûteuses) incompatibles avec les caractéristiques de l'informatique mobile, qui se distingue en particulier par le nomadisme des utilisateurs accédant depuis n'importe quelle localisation géographique à leurs applications. En dépit de tous les problèmes mentionnés ci-dessus, un utilisateur mobile souhaite se déplacer librement et continuer à travailler le plus normalement possible avec son terminal mobile. Il est donc souhaitable de

fournir une continuité de service malgré les déconnexions et les perturbations du réseau sans fil.

Le besoin de continuer à travailler dans un environnement mobile soulève de nombreux problèmes. Tous d'abord, l'approche visant à résoudre la disponibilité passe par une réplication des données et du code sur le terminal mobile. En outre, d'après [Satyanarayanan, 1996a], l'adaptation aux caractéristiques de l'informatique mobile est exprimable en trois stratégies : « laissez-faire » où l'adapta-tion est entièrement de la responsabilité de l'application, donc sans aucun support du système ; « transparence » où aucun changement de l'application n'est nécessaire puisque c'est le système qui est responsable de l'adaptation ; « collaboration » où l'adaptation est faite en collaboration entre l'application et le système. De nombreux travaux synthétisés dans [Jing *et al.*, 1999] montrent que les approches « laissez-faire » et « transparence » ne sont pas adéquates. Ainsi, dans nos travaux, nous nous basons sur la stratégie d'adaptation « collaboration ».

La réplication des données sur le terminal mobile consiste à instancier le même type d'entité sur le terminal mobile que sur le serveur fixe où elle se situe normalement. Ces entités déconnectées vont être utilisées pour assurer la continuité de service lors de déconnexions. Les opérations effectuées pendant les phases de déconnexion sont journalisées localement pour être envoyées lors de la reconnexion aux serveurs pour effectuer la réconciliation. Ce processus de traitement des déconnexions encapsule le déploiement des entités distantes sur le terminal mobile. La plupart des algorithmes de déploiement existant dans la littérature traitent ce problème selon un aspect quantitatif : nombre d'entités en mémoire et l'élection des entités que le système supprime en cas de dépassement de la taille mémoire. Cependant, peu de travaux traitent l'aspect qualitatif de l'entité susceptible d'être déployée sur le terminal mobile. C'est sur ce dernier aspect que se penche notre travail dans cet article où nous proposons deux métadonnées : « nécessité » et « déconnectabilité ». Ces deux méta-données répondent à deux questions. Premièrement, avoir des mandataires dans le terminal mobile augmente la disponibilité du service, mais est-ce que toutes les entités distantes peuvent avoir une entité déconnectée (déconnectabilité) ? Deuxièmement, est-ce que ces entités déconnectables sont nécessaires pour le fonctionnement de l'application en mode déconnecté ?

Dans le méta-modèle que nous présentons dans la section 3, nous utilisons le mot « entité » dans l'acception « granularité de l'entité manipulée par le système » (par exemple, « fichier » dans les systèmes orientés fichiers, « enregistrement » dans les systèmes de bases de données, « objet » dans les systèmes orientés objets et « composant » dans les systèmes orientés composants). Dans la section 5, nous présentons l'architecture D✳MINT avec des composants CORBA. Nous avons choisi CCM pour son interopérabilité, son utilisation dans des domaines variés et sa portabilité sur des assistants personnels numériques. Les résultats peuvent cependant être appliqués à d'autres modèles de composants.

3 - MÉTA-MODÈLE POUR LA GESTION DE LA CONNECTIVITÉ

La gestion de la connectivité dépend largement de la sémantique de l'application. Cette sémantique peut être modélisée par des données appelées méta-données. Dans cette section, nous utilisons les méta-données, d'une part, pour spécifier si une entité distante peut avoir une entité déconnectée sur le terminal mobile (cf. Section 3.1), d'autre part, pour spécifier si cette entité déconnectée est nécessaire pour l'exécution de l'application en mode déconnecté (cf. Section 3.2). Pour mieux comprendre les méta-données que nous définissons dans cette section, nous illustrons leur utilisation avec une application de messagerie électronique que nous avons implantée. Cette application comporte trois types d'entités distribuées : l'entité `MailBoxManager` responsable de la création, de la suppression et de la localisation des entités `MailBox` ; ces dernières sont responsables de l'envoi, de la réception et de la suppression des messages ; enfin, l'entité `AddressBook` fournit un outil de consultation, d'ajout et de suppression d'adresses des utilisateurs.

3.1 - Méta-donnée déconnectabilité

La méta-donnée « déconnectabilité » indique si cette entité présente dans le terminal fixe accepte une entité déconnectée dans le terminal mobile. Ainsi, les entités distribuées de l'application sont partitionnées en deux groupes. La figure 1 présente un patron de conception pour le partitionnement des entités de l'application, plus l'attribution de la méta-donnée « nécessité » présentée dans la section 3.2. Le partitionnement en entités déconnectables et entités non déconnectables est réalisé par le développeur[1] de l'application qui doit concevoir son application suivant ces contraintes. Ce partitionnement dépend de la sémantique de l'application. Par exemple, pour des raisons de sécurité, le développeur de l'application

[1] Dans le processus de développement d'applications à base de composants CORBA, le développeur de l'application peut être décomposé en concepteur du composant, concepteur de composition de composants, implanteur du composant, empaqueteur du composant et déployeur du composant.

Entité	Déconnectabilité	Nécessité		
		Choix du programmeur	Choix de l'utilisateur	Finale
MailBox	Oui	EN	n/a	EN
MailBoxManager	Non	n/a	n/a	n/a
AddressBook	Oui	ENN	EN	EN

TAB. **1.** Partitionnement des entités de l'application de messagerie électronique : n/a signifiée « non applicable ».

peut décider de déployer des entités de l'application dans des serveurs sécurisés. Ces entités ne peuvent

déconnectable. Par contre, les entités `MailBox` et `AddressBook` peuvent avoir une entité déconnectée dans le terminal

création des entités déconnectées pour ces entités n'est pas autorisée. Les entités déconnectées doivent être conçues pour faire face à une future réconciliation avec les entités distantes. Dans cet article, le problème de la réconciliation ne sera pas abordé. Ce problème est abordé dans [Chateigner *et al.*, 2003].

La seconde colonne de la table 1 présente l'affectation de la méta-donnée pour les entités de l'application de messagerie électronique. Dans cette application, supposons que le client administrateur de l'application crée dans son terminal mobile un mandataire pour l'entité `MailBoxManager`.

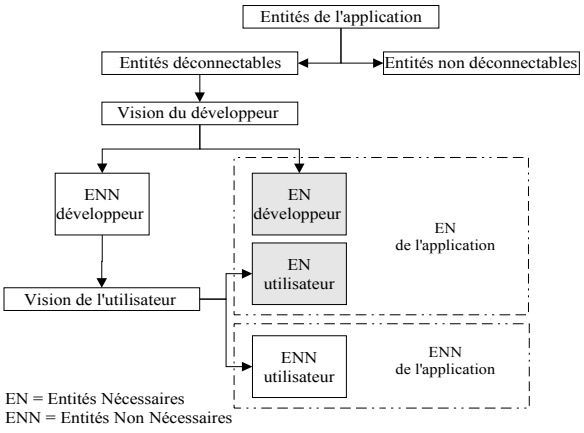

FIG. **1.** Méta-modèle pour la partitionnement des entités de l'application.

Dans le scénario où l'administrateur peut manipuler cet objet en mode déconnecté et créer une nouvelle boîte aux lettres pour un autre utilisateur, ce dernier utilisateur ne pourra pas utiliser sa boîte aux lettres tant que l'administrateur est déconnecté et que la réconciliation n'a pas eu lieu. Donc, l'entité `MailBoxManager` peut être classée par le développeur de l'application comme étant une entité non

veut lire, envoyer ou supprimer des messages. Les messages manipulés par l'utilisateur dans l'entité déconnectée sont les messages téléchargés à partir de l'entité dans le serveur fixe avant la déconnexion. Ce mécanisme permet quand même à l'entité restée dans le réseau fixe de recevoir les messages à destination de l'utilisateur. Cet utilisateur verra ses messages lors de la reconnexion et les messages qu'il a envoyés à partir de l'entité déconnectée seront alors acheminés vers les destinataires. L'entité `AddressBook` est personnelle à un utilisateur, c'est une entité passive. Ainsi, si l'utilisateur ajoute ou supprime des entrées dans sont carnet d'adresses, ces changements n'auront pas un impact sur d'autres entités partagées de l'application. Cette entité est donc déconnectable.

3.2 - Méta-donnée nécessité

La méta-donnée que nous avons présentée dans la section précédente ne traite pas le problème de la quantité d'entités déconnectées devant co-exister dans le faible espace mémoire du terminal mobile. Pour résoudre ce problème, nous introduisons une autre méta-donnée : « nécessité», qui permet de donner un « poids » aux entités de l'application quant à leur présence dans le terminal mobile. Ce concept nous amène à classer les entités déconnectables en entités nécessaires (EN) et entités non nécessaires (ENN). Une entité nécessaire est une entité dont la présence dans le terminal mobile est obligatoire pour le fonctionnement en mode déconnecté. Une entité non nécessaire est une entité dont l'absence dans le terminal mobile ne peut pas empêcher le fonctionnement de l'application en mode déconnecté. L'application doit être construite de telle façon qu'une telle entité absente pendant une déconnexion n'empêche pas l'application de continuer à fonctionner. Comme présentée dans la figure 1, la classification des entités de l'application en EN et ENN est effectuée d'abord par le développeur et ensuite par l'utilisateur. Le développeur de l'application fournit la première classification en « EN développeur » et « ENN développeur ». Ensuite, à chaque lancement de

l'application ou dans la configuration de cette dernière à l'installation, l'utilisateur peut forcer un « ENN développeur » à devenir nécessaire, une telle entité étant alors appelée « EN utilisateur ». Dans la figure 1, les rectangles grisés représentent les EN de l'application, c'est-à-dire l'union des « EN développeur » et des « EN utilisateur ». En outre, l'utilisateur ne peut pas forcer un « EN développeur » à devenir un « ENN utilisateur ». Nous considérons que le développeur connaît mieux que l'utilisateur quelles sont les entités déconnectées qui doivent impérativement exister dans le terminal mobile pour le fonctionnement en mode déconnecté.

La troisième colonne de la table 1 présente l'affectation de la méta-donnée « nécessité » pour les entités de l'application de messagerie électronique. Il est clair que la présence d'un mandataire pour `MailBox` est nécessaire pour le fonctionnement de l'application. Par contre, `AddressBook` est classée comme ENN par le développeur puisque l'utilisateur peut se passer de son carnet d'adresses et n'envoyer des messages qu'aux destinataires dont il connaît l'adresse. Lorsque l'utilisateur lance son application de messagerie électronique sur son terminal mobile ; il s'aperçoit qu'il a le droit de modifier la nécessité de AddressBook. S'il pense qu'il va envoyer plusieurs messages à des destinataires différents et qu'il ne connaît pas par cœur leur adresse, il peut forcer la nécessité de AddressBook.

4 - LE MODÈLE DE COMPOSANT CCM

L'utilisation de modèles et langages orientés objets pour la construction des applications réparties existe depuis longtemps. L'objectif de ces recherches est d'améliorer la modélisation d'une application, d'optimiser la réutilisation du code métier et d'adresser l'ensemble du cycle de développement. Cependant, la conception orientée objet n'est pas adaptée à la description de schémas de coordination et de communication complexes. Pour pallier les limites de l'approche orientée objets et adresser des notions non prévues initialement, l'approche composant est apparue. Dans cette section, nous présentons d'abord le modèle de composant CCM (cf. Section 4.1), ensuite nous discutons l'utilisation des méta-données présentées dans la section 3 dans le contexte CCM (cf. Section 4.2).

4.1 - Vue d'ensemble de CCM

Il n'existe pas, aujourd'hui, une définition unique du « composant logiciel ». Toutefois, plusieurs définitions ou caractéristiques sont acceptées. Szyperski [Szyperski *et al.*, 2002] définit un composant comme *« une unité de composition avec des interfaces contractuellement spécifiées et des dépendances explicites sur son contexte. Un composant peut être déployé indépendamment et il est sujet à des compositions par tiers. »*

Dans la suite de cet article, notre raisonnement est illustré dans le cadre de CCM dont des implantations existent déjà : par exemple, OpenCCM [Marvie et Merle, 2001] et micoCCM [Pilhofer, 2002]. Le choix de CCM est justifié par plusieurs points. Premièrement, par rapport aux autres modèles tels que EJB [DeMichiel, 2002], COM [Rogerson, 1997] et .NET [Microsoft, 2002], CCM peut être vu comme l'union de ces modèles. Comme dans EJB, un composant CORBA est créé et géré par une maison et exécuté dans un conteneur. Comme dans COM, un composant CORBA offre et utilise des interfaces et il permet la navigation et l'introspection. Comme dans .NET, un composant CORBA peut être écrit dans plusieurs langages de programmation et peut être empaqueté pour être distribué. Deuxièmement, CCM est multilanguages, multiOS, multiORB, multivendeur, contrairement à EJB qui est purement JAVA, et COM et .NET qui sont purement Windows.

Un composant CCM est caractérisé par des ports classés en synchrones (facettes et réceptacles) et asynchrones (sources et puits d'événements), et des attributs pour la configuration (cf. figure 2). Un autre avantage avec CCM est que le composant peut être segmenté (cf. figure 3).

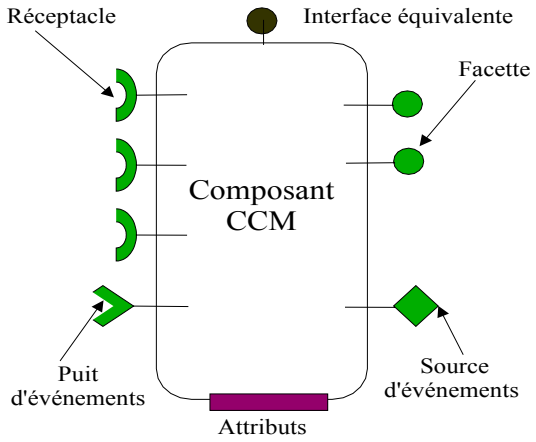

FIG. **2.** Vision externe d'un composant CCM.

Pour chaque segment, CCM génère un squelette. Les segments sont activés indépendamment et possèdent un état. Enfin, chaque segment est séparément identifié dans le système.

4.2 - Déconnectabilité et nécessité dans CCM

Dans cette section, nous présentons juste une brève discussion sur l'utilisation des méta-données « déconnectabilité » et « nécessité » dans le contexte CCM. Ce point a été développé dans [Kouici *et al.*, 2003]. Un composant déconnectable est un composant qui peut avoir un composant déconnecté dans le terminal mobile utilisé en mode déconnecté. Dans le cas d'une implantation monolithique du composant, tous les services offerts par le composant sont implantés dans une seule classe. Si le composant est déconnectable, le segment qui correspond à la totalité du composant est aussi déconnectable.

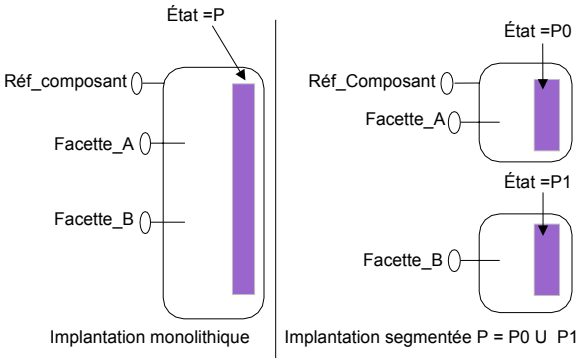

FIG. **3.** Composants monolithique et segmenté.

Dans le cas d'une implantation segmentée, ce mandataire ne contient que des segments déconnectables. En outre, un composant déconnectable contient au moins un segment déconnectable et s'il existe un segment déconnectable alors ce composant est déconnectable.

CCM définit la notion de segment principal (*main segment*). Ce segment contient tous les services et attributs non affectés à d'autres segments. Ce segment possède aussi l'exclusivité de manipulation du contexte par réflexion. En particulier, il offre la possibilité de localiser les autres segments par l'interface `ExecutorLocator`. Ainsi, il est clair que la déconnectabilité du composant implique la déconnectabilité de son segment principal.

Nous définissons aussi la notion de composant et de segment nécessaires par analogie avec la déconnectabilité. Un composant nécessaire doit contenir au moins un segment nécessaire et si un segment est déclaré nécessaire alors son composant est aussi nécessaire. La localisation des services et des segments étant nécessaire en mode déconnecté, la nécessité d'un composant implique la nécessité de son segment principal.

5 - ARCHITECTURE DE D✳MINT

La première partie de cette section décrit l'intégration des méta-données présentées dans la section 4.2 dans la plate-forme OpenCCM. Ensuite, nous présentons brièvement l'architecture D✳MINT.

OpenCCM est une plate-forme écrite en Java. Elle fournit une première implantation partielle de CCM. CCM étend le langage OMG IDL pour permettre d'exprimer les caractéristiques du composant décrites dans la section 4.1. En plus, il définit un nouveau langage, *Component Implementation Definition language* (CIDL), pour spécifier l'architecture interne des composants. Nous avons ajouté dans le CIDL de nouvelles entrées grammaticales pour spécifier la déconnectabilité et la nécessité d'abord pour le composant, c-à-d. pour le segment principal, ensuite pour les autres segments du composant. Nous avons aussi ajouté des étapes dans la chaîne de compilation d'OpenCCM [Flissi, 2003] pour inclure dans chaque squelette du segment le code nécessaire pour spécifier la déconnectabilité et la nécessité.

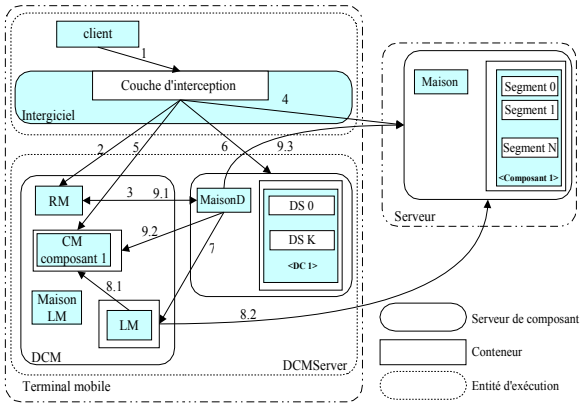

FIG. **4.** Architecture de D✳MINT.

La figure 4 représente l'architecture de D✳MINT. D✳MINT crée pour chaque terminal mobile un service de gestion des déconnexions `DCMServer` qui se trouve dans une entité d'exécution indépendante de celles des applications dans le terminal mobile. Le `DCMServer` est accessible par l'intergiciel de l'application via une couche d'interception propre à chaque instance de l'intergiciel. Dans CORBA, cette couche d'interception est pilotée par des objets CORBA locaux appelés les intercepteurs portables [Object Management Group, 2001a]. Cette couche intercepte toutes les requêtes du client vers les serveurs. Le gestionnaire des composants déconnectés (DCM) comporte une fabrique pour les composants/segments déconnectés (RM) et un

gestionnaire du journal d'opérations (LM). À chaque composant déconnecté est associé un gestionnaire de connectivité (CM).
La couche d'interception détient une table qui fait le lien entre une référence de segment et un gestionnaire de connectivité. Imaginons le scénario où D✳MINT doit créer un segment/composant déconnecté pour chaque segment/composant à la première invocation. Lorsqu'un client invoque un service du composant (via le segment associé) (1), la couche d'interception intercepte cette requête et contrôle la déconnectabilité de ce segment. Dans le cas où ce segment accepte un mandataire, la couche d'interception contrôle si le composant déconnecté associé à ce segment existe déjà dans DCMServer. La requête étant la première pour ce composant, la couche d'interception obtient la référence de RM (2). RM crée d'abord une maison déconnectée (3) utilisée pour la création du composant déconnecté (DC). Ensuite, RM crée le gestionnaire de connectivité (CM) associé et le segment déconnecté (DS) pour le segment invoqué. Au moment de la création de DC/DS, l'intergiciel transmet la requête du client vers le composant distant (4).
Les requêtes du client sont interceptées par la couche d'interception. Cette dernière contrôle la connectivité via le CM associé au segment invoqué (5). Si la connectivité est bonne, la requête est dirigée vers le composant distant (4). Sinon, la requête est dirigée vers le DS/DC associé (6). Les opérations effectuées sur DC/DS vont être sauvegardées dans LM (7). Périodiquement, LM teste la connectivité (8.1), et si possible, envoie les requêtes au composant distant (8.2). La maison déconnectée comporte du code nécessaire pour chercher la référence de CM (9.1), tester la connectivité (9.2) et faire un transfert d'état entre DC et le composant distant (9.3). Le mécanisme de réconciliation entre l'entité distante et les entités déconnectées est étudié dans [Chateigner *et al.*, 2003].

6 - TRAVAUX CONNEXES

La problématique traitée dans cet article aborde les domaines de recherche autour de la mobilité des applications (gestion de la déconnexion), des méta-données (configuration par réflexion) et des modèles de composants.
Un panorama est donné dans [Jing et al., 1999] sur l'adaptation à la mobilité des applications client/serveur, qui se concentre de plus en plus sur la gestion et le déploiement[2] des entités déconnectées dans le terminal mobile. Coda [Satyanarayanan, 1996b, Satyanarayanan, 1996a] 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 construite par le client de l'application. Si l'utilisateur fait un mauvais choix pour ses données explicites suite à une mauvaise compréhension de l'application, cette dernière peut ne pas fonctionner en mode déconnecté. Odyssey [Mummert, 1996, Noble et Satyanarayanan, 1999] 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. Rover [Joseph *et al.*, 1997] définit les notions d'objet dynamique relogeable (*Relocatable Dynamic Object, RDO*) et d'appel de procédure à distance non-bloquant (*Queued Remote Procedure Call, QRPC*). Rover traite tous les objets de l'application de la même façon et ne tient pas compte de la sémantique de l'application. En outre, le développeur doit concevoir son application en terme de RDO.
L'usage des méta-données a été aussi introduit dans les intergiciels réflexifs [Blair et al., 1998, Parlavantzas et al., 2000]. XMIDDLE [Capra et al., 2001] définit la méta-donnée « profil » qui décrit ce que l'intergiciel doit faire lorsqu'il se trouve dans un contexte particulier. Cette méta-donnée contient des informations sur les ressources externes (bande passante, batterie…) et ne prend pas en compte la sémantique de l'application. Ce type de méta-donnée a été aussi introduit dans CARISMA [Capra et al., 2003], qui associe à chaque application un fichier de description de comportements.
Dans le contexte CORBA, les projets Π^2 [Ruggaber *et al.*, 2000] et Alice [Lynch, 1999] traitent le problème des courtes déconnexions provoquées par le changement de cellule (*handover*) dans *wireless* CORBA [Object Management Group, 2001b]. Π^2 ne traite que le problème des déconnexions involontaires et utilise deux objets déconnectés, un sur le terminal mobile et l'autre sur le terminal fixe. Alice fournit des mécanismes pour traiter les déconnexions volontaires et involontaires en utilisant les exceptions produites par l'intergiciel lors de changements de cellules. Dans Alice, le code nécessaire pour commuter vers le mode déconnecté doit être inclus dans le code de l'application. Dans notre approche, l'utilisation d'une couche d'interception au niveau de l'intergiciel permet de limiter l'impact sur le code de l'application. CASCADE [Chockler *et al.*, 2000, Atzmon *et al.*, 2002] définit un service de gestion des mandataires générique et hiérarchique pour des objets CORBA. Il définit un service de désignation pour les serveurs de mandataires et les clients utilisent ce

[2] Ici, le mot déploiement signifie le téléchargement ou la création des entités déconnectées sur le terminal mobile.

service pour trouver des serveurs où créer des mandataires. Dans CASCADE, si le client se déconnecte avec tous les serveurs comportant les mandataires, il ne peut pas continuer à travailler si sa machine n'est pas elle-même un serveur de mandataires.
Enfin, dans le domaine des composants, CESURE [Marangozova et Hagimont, 2002] traite le problème de la réplication des composants d'une manière extra-fonctionnelle. Ce projet introduit la notion de composant de déconnexion qui représente une réplique du composant distant, et qui contient en plus le code nécessaire pour la réconciliation. L'approche utilisée dans CESURE ne traite pas le problème des déconnexions involontaires puisque les mandataires sont créés sur des hôtes autres que le terminal mobile.

7 - CONCLUSION

Dans cet article, nous avons présenté l'état actuel de nos travaux sur l'adaptation des applications réparties en environnements mobiles. Nous avons proposé un patron de conception pour des applications réparties fonctionnant dans des environnements mobiles. Ce patron de conception se base sur l'utilisation des méta-données « déconnectabilité » et « nécessité ». La méta-donnée « déconnectabilité » permet de spécifier si une entité dans un serveur fixe peut avoir un mandataire dans le terminal mobile. La méta-donnée « nécessité » permet de donner un « poids » à l'entité déconnectée quant à sa présence dans le terminal mobile. Ces méta-données permettent de créer un lien de collaboration entre le développeur de l'application et l'utilisateur de cette dernière.
En utilisant ces méta-données et en s'appuyant sur la plate-forme Domint, nous avons présenté D✳MINT. Nous montrons avec cette architecture que la collaboration pour le traitement de la mobilité ne s'arrête pas à l'utilisation des méta-données, mais demande aussi l'utilisation d'autres ressources systèmes tels que les intercepteurs au niveau de l'intergiciel.
Les méta-données que nous avons présentées dans cet article ne traitent pas le problème de la priorité entre les différentes entités de l'application. Cette priorité doit être spécifier dans une autre méta-donnée : « criticité ». Les autres perspectives de nos travaux sont les suivantes. Tout d'abord, nous allons terminer l'implantation de D✳MINT, porter cette architecture sur des terminaux mobiles (PDA) et faire des tests de performances. L'architecture actuelle de D✳MINT utilise des composants métier. Cependant, la gestion de la mobilité doit être vue comme un aspect extra-fonctionnel géré par le conteneur lui-même. Donc, une perspective supplémentaire est l'intégration de D✳MINT dans les conteneurs CCM en utilisant des composants système. Une approche prometteuse est celle des conteneurs ouverts [Vadet et Merle, 2001].
Enfin, D✳MINT ne prend en compte que les invocations synchrones. Or, les communications asynchrones deviennent de plus en plus utilisées dans les intergiciels via des services d'événements ou de notification. Le modèle de composant CCM permet les deux types de communication. Nous travaillons donc à étendre D✳MINT dans ce sens.

BIBLIOGRAPHIE

[Atzmon *et al.*, 2002] Atzmon H., Friedman R. et Vitenberg R. (2002). Replacement Policies for a Distributed Object Caching Service. Dans *Proceedings of the International Symposium on Distributed Objects and Applications*, pages 661–674, California, Irvine, USA.

[Blair *et al.*, 1998] Blair G. S., Coulson G., Robin P. et Papathomas, M. (1998). An Architecture for Next Generation Middleware. Dans *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, London, England

[Capra *et al.*, 2001] Capra L., Emmerich W. et Mascolo C. (2001). Exploiting Reflection and Metadata to built Mobile Computing Middleware. Dans *Proceedings of the Workshop on Middleware Mobile Computing, IFIP/ACM*, Heidelberg, Germany.

[Capra *et al.*, 2003] Capra L., Emmerich W. et Mascolo C. (2003). CARISMA : Context-Aware Reflective Middleware System for Mobile Applications. *IEEE Transaction on Software Engineering*.

[Chateigner *et al.*, 2003] Chateigner L., Chabridon S. et Bernard G. (2003). Intergiciel pour l'informatique nomade : réplication optimiste et réconciliation. Dans Actes de la *Manifestation des Jeunes Chercheurs STIC, (MAJECSTIC)*, Marseille, France.

[Chockler *et al.*, 2000] Chockler G., Dolev D., Friedman R. et Vitenberg R. (2000). Implementing a caching service for distributed CORBA objects. Dans *Prceedings of the 2nd FIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 1–23.

[Conan *et al.*, 2002a] Conan D., Chabridon S.,Villin O. et Bernard G. (April 2002a). Disconnected Operations in Mobile Environments. Dans *Proceedings of the. 2nd IPDPS Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, Ft. Lauderdale, USA.

[Conan *et al.*, 2002b] Conan D., Chabridon S., Villin O., Bernard G., Kotchanov A. et Saridakis T. (2002b). Handling Network Roaming and Long Disconnections at Middleware Level. Dans *Proceedings of the Workshop on Software Infrastructures for Component-Based applications on Consumer Devices*, Lausanne, Switzerland.

[DeMichiel, 2002] DeMichiel L. (2002). *Enterprise JavaBeans Specifications, version 2.1, proposed final draft*. Sun Microsystems, http ://java.sun.com/products/ejb/docs.html.

[Flissi, 2003] Flissi A. (2003). Inside OpenCCM. Rapport technique, ObjectWeb.

[Jing *et al.*, 1999] Jing J., Helal A. et Elmagarmid A. (June 1999). Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2).

[Joseph *et al.*, 1997] Joseph A., Tauber J. et Kaashoek M. (1997). Mobile computing with the Rover toolkit. *ACM Transactions on Computers*, 46(3).

[Kouici *et al.*, 2003] Kouici N., Conan D. et Bernard G. (2003). Disconnected Metadata for Distributed Applications In Mobile Environments. Dans *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, USA.

[Lynch, 1999] Lynch N. (1999). Supporting Disconnected Operation in Mobile CORBA. Thèse de doctorat, Trinity College Dublin.

[Marangozova et Hagimont, 2002] Marangozova V. et Hagimont D. (2002). An Infrastructure for CORBA Component Replication. Dans *Proceedings of the First International IFIP/ACM Working Conference on Component Deployment*, Berlin (Germany).

[Marvie et Merle, 2001] Marvie R. et Merle P. (2001). CORBA Component Model: Discussion and Use with OpenCCM. Rapport technique, Laboratoire d'Informatique Fondamentale de Lille, France.

[Microsoft, 2002] Microsoft (2002). Microsoft Developer Network. http ://www.msdn.microsoft.com.

[Mummert, 1996] Mummert L. (September 1996). Exploiting Weak Connectivity in a Distributed File System. Thèse de doctorat. Université de Carnegie Mellon.

[Noble et Satyanarayanan, 1999] Noble B. D. et Satyanarayanan M. (1999). Experience with Adaptive Mobile Applications in Odyssey. *Mobile Networks and Applications*, 4(4) page 245–254.

[Object Management Group, 2001b] Object Management Group (June 2001b). Wireless Access and Terminal Mobility in CORBA Specification. OMG Document dtc/01-06-02.

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

[Object Management Group, 2001a] Object Management Group (September 2001a). Portable Interceptors. Interceptors Finalization Task Force. Published draft.

[Parlavantzas *et al.*, 2000] Parlavantzas N., Coulson G., Clarke M. et Blair G. (2000). Towards a Reflective Component-based Middleware Architecture. Dans *Proceedings of the Workshop on Reflection and Metalevel Architectures*, Sophia Antipolis et Cannes, France.

[Pilhofer, 2002] Pilhofer F. (2002). Writing and Using CORBA Component. Rapport technique, ALCATEL. http ://www.fpx.de/MicoCCM/.

[Rogerson, 1997] Rogerson D. (1997). *Inside COM*. Microsoft Press.

[Ruggaber et al., 2000] Ruggaber R., Seitz J. Et Knapp M. (July 2000). Π^2 A Generic Proxy Platform for Wireless Access and Mobility. Dans *Proceedings of the 19th ACM*

Symposium on Principles of Distributed Computing, Portland, Oregon.

[Satyanarayanan, 1996a] Satyanarayanan M. (1996a). Fundamental Challenges in Mobile Computing. Dans *Proceedings of the 15th Symposium on Principles of Distributed Computing*, pages 1–7.

[Satyanarayanan, 1996b] Satyanarayanan M. (1996b). Mobile Information Access. *IEEE Personal Communications*, 3(1).

[Szyperski et al., 2002] Szyperski C., Gruntz D. et Murer S. (2002). *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley.

[Vadet et Merle, 2001] Vadet M. et Merle P. (2001). Les conteneurs ouverts dans les plates-formes à composants. Dans *Actes de la Journée Thème Émergent Composants*, Besançon, France.