Domint*: une plate-forme pour la faible connectivité et la déconnexion d'objets CORBA en environnement mobile

Denis Conan, Sophie Chabridon, Olivier Villin et Guy Bernard
GET / INT, CNRS Samovar, 9 rue Charles Fourier, 91011 Évry, France
{Denis.Conan, Sophie.Chabridon, Olivier.Villin, Guy.Bernard}@int-evry.fr

May 2003

Résumé

Les caractéristiques intrinsèques des communications sans fil restent le facteur principal limitant les performances des applications mobiles. De nouvelles approches au niveau des intergiciels doivent apparaître pour masquer ces limitations ainsi que pour faciliter le développement d'applications mobiles. Cet article présente Domint, une plate-forme permettant d'adapter des applications réparties CORBA à un environnement mobile sujet à des déconnexions ou souffrant d'une faible connectivité. Un objet mandataire de l'objet serveur distant appelé « objet déconnecté », est automatiquement déployé sur le terminal mobile et la gestion de la connectivité repose sur un mécanisme d'hystérésis évitant de trop fréquents transferts d'états ou de journaux, et de trop fréquentes commutations entre l'objet déconnecté et l'objet serveur distant. La commutation entre l'objet déconnecté et l'objet serveur distant est obtenue par l'utilisation des intercepteurs portables CORBA. L'article montre les résultats de performances d'un prototype fonctionnant sur PC (avec Windows 2000 ou Redhat Linux) et sur PDA iPAQ (avec Windows CE ou Linux Familiar).

1 Introduction

Les récentes avancées dans le domaine des communications sans fil et les progrès dans les terminaux portables ont rendu possibles de nouvelles applications dans lesquelles l'usager peut avoir accès à l'information à n'importe quel moment et depuis n'importe où. La société de l'information de demain bénéficiera de la mobilité qui deviendra la règle et non plus l'exception dans des environnements saturés de moyens de calculs et de communication au service des usagers [26].

*Domint est un projet Open Source sous licence GNU GPL et les fichiers sources sont disponibles sur https ://picolibre.int-evry.fr/.

Le développement d'applications mobiles est facilité par la technologie des intergiciels. Un standard pour l'accès sans fil et la mobilité des terminaux dans les applications réparties a été défini par l'Object Management Group (OMG) en 2001 [19]. Cette spécification prend en compte la mobilité des usagers au niveau de l'intergiciel et permet de tolérer les déconnexions réseau transitoires survenant lors d'un changement de cellule au moyen d'un protocole de redirection des messages. Cependant, les applications mobiles sont également sujettes à des déconnexions réseau de longue durée et nécessitent la mise en place de mécanismes supplémentaires.

Notre principale contribution est la réalisation de la plate-forme Domint pour l'adaptation d'applications CORBA légataires à un fonctionnement en environnement mobile en assurant la continuité de service en cas de faible connectivité, voire de déconnexion complète. Une faible connectivité résulte d'une connexion intermittente, d'une bande passante réduite, d'une forte latence ou d'une communication réseau onéreuse [14]. Nous distinguons deux types de déconnexions : les déconnexions volontaires sont à l'initiative de l'usager pour économiser la batterie ou les coûts de communication par exemple, et les déconnexions involontaires résultent d'une rupture de la connexion physique comme lorsque la station de base du réseau est hors de portée. La sémantique de l'application en environnement mobile est différente de celle qu'elle aurait en un environnement filaire. Nous considérons cette différence comme acceptable à condition que l'usager soit informé en permanence de l'état de la connexion, via une interface graphique adaptée par exemple. Nous ne traitons pas ici de la cohérence des données ni de la réconciliation de données divergentes survenant naturellement lors de déconnexions et reconnexions successives.

Le reste de cet article présente l'architecture de la plate-forme Domint, son implémentation et l'évaluation de ses performances. La section 2 développe les choix de conception et décrit l'architecture. La gestion de la connectivité est détaillée à la section 3. La section 4 décrit la manière transparente dont sont effectués les changements de mode de travail en fonction de la connectivité tandis que la section 5 traite de la conception des objets déconnectés et des interactions avec le mécanisme de journalisation. La section 6 évalue les performances de Domint sur PC (avec Windows 2000 ou Redhat Linux) et sur PDA iPAQ (avec Windows CE ou Linux Familiar) pour une application de messagerie électronique. Dans la section 7, nous donnons un état des travaux de recherche connexes et nous concluons dans la section 8.

2 Motivations, objectifs et architecture de Domint

Dans une application répartie « classique » fonctionnant avec une bonne connectivité entre les différents composants, les clients peuvent s'exécuter sur des terminaux légers et n'être constitués que d'une interface graphique, les objets serveurs restant sur des nœuds du réseau fixe. Pour y ajouter la continuité de service lors de déconnexions, des mandataires des objets serveurs doivent être instanciés sur les hôtes des clients, les opérations pendant les phases de déconnexion journalisées et des réconciliations lors des reconnexions opérées. Cette section présente tout d'abord les motivations et les

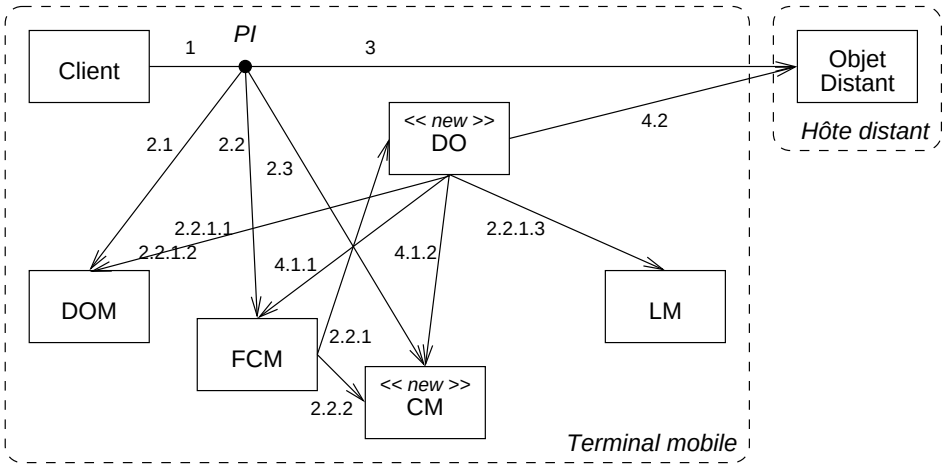
objectifs de Domint dans la section 2.1 puis son architecture en détail dans la section 2.2.

2.1 Motivations et objectifs

La faible connectivité des environnements mobiles ajoutée à la relative pauvreté en ressources des terminaux mobiles amène aux deux compromis suivants : systèmes non interopérables liés à un matériel dédié contre systèmes généraux interopérables mettant à disposition de nombreux services variés ; applications autonomes contre applications réparties constituées de composants inter-dépendants.

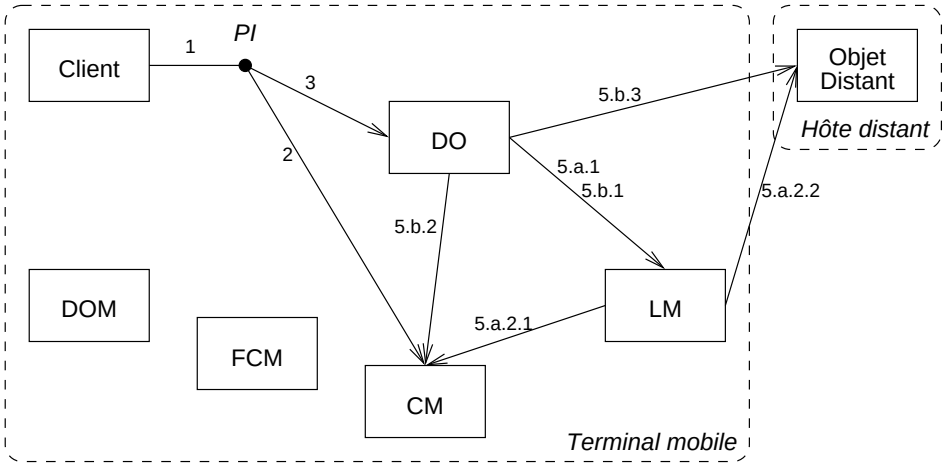
Même dans le monde CORBA qui fournit l'interopérabilité de façon native, le premier compromis est illustré par la présence de la spécification d'une version minimale de CORBA [17] comme un sous-ensemble de la spécification complète incluant de nombreux services généraux, facilités et services liés à un métier. Nous avons choisi CORBA justement pour son utilisation dans des domaines variés et aussi pour ses mécanismes d'extensibilité tels que les intercepteurs portables permettant l'ajout transparent de services extra-fonctionnels (*cf*. Section 4). Le prototype présenté en section 6 montre que les assistants numériques personnels existants et *a fortiori* les matériels futurs peuvent contenir notre plate-forme, qui contient un ORB (« *Object Request Broker* ») complet. En outre, la continuité de service pendant les déconnexions est assurée par le chargement d'objets CORBA mandataires appelés objets déconnectés, similaires en conception et en implantation aux objets restants sur le réseau fixe, mais spécifiquement construits pour faire face au fonctionnement en mode déconnecté. 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 objets déconnectés étant des objets CORBA, d'une part, ne sont pas locaux à une application, mais partageables par toutes les applications s'exécutant sur le terminal mobile, d'autre part, peuvent utiliser les services standards de CORBA indépendamment de Domint.

Le compromis entre l'autonomie et l'inter-dépendance des composants d'une application répartie est exprimable en trois stratégies [25] : « laissez-faire » ou aucun support du système mais adaptation de l'application ; collaboration entre le système et l'application ; « transparence », ou aucun changement de l'application et support transparent du système. De nombreux travaux [11, 12, 15, 22, 29] ont montré que le système seul peut rester efficient même lorsque la bande passante du réseau de communication varie de plusieurs ordres de grandeur, mais aussi que le système requiert l'intervention de l'application pour améliorer son agilité (vitesse et précision) et pour spécifier la fidélité en terme de cohérence des données. Ainsi, dans nos travaux, afin de gérer plusieurs applications de façon concomitante sur le même terminal mobile, certaines parties de la gestion des ressources et de la journalisation sont imparties au système de manière transparente. Ces services sont rendus par des objets CORBA configurables par les applications.



1: requête du Client avant l'interception
2.1: findFCM() *2.2: findCM()* *2.2.1: createDO()* *2.2.1.1: findFCM()*
2.2.1.2: findLM() *2.2.1.3: addInterpretLogOBV()* *2.2.2: createCM()* 2.3: getCMInfo()
3: requête du Client après l'interception
4.1.1: findCM() *4.1.2: getCMInfo()* 4.2: <<périodique>> incrementalStateTransfer()

(a)



1: requête du Client avant l'interception
2: getCMInfo()
3: requête du Client après l'interception
4: getCMInfo()
5.a.1: addLog() 5.a.2.1: << periodic >> getCMInfo() 5.a.2.2: requête du DO
5.b.1: treatLog() 5.b.2: getCMInfo() *5.b.3: requête du DO*

(b)

FIG. 1 – L'architecture de Domint.

2.2 Architecture de Domint

L'architecture de Domint est présentée Figure 1. Les deux dessins présentent les diagrammes de collaborations UML d'un client émettant une première requête vers un objet distant alors que la connectivité est forte, puis émettant une autre requête alors que la connectivité est devenue faible, respectivement. Dans cette section, les différentes enti-

tés de Domint sont présentées dans la section 2.2.1 et les diagrammes de collaborations commentés dans les sections 2.2.2 et 2.2.3.

2.2.1 Rôle des entités

Tous les rectangles de la figure 1 représentent des objets CORBA. L'intercepteur portable dénoté *PI* est aussi un objet CORBA, mais local, c.-à-d. qu'il ne peut pas être adressé hors de l'entité d'exécution l'englobant[1]. Toutes les requêtes et les réponses du *Client* sont interceptées par *PI*. Pour les requêtes sortantes, *PI* agit comme un commutateur entre l'objet déconnecté dénoté *DO* et l'*Objet Distant*. Lors de la réception des réponses, *PI* détecte les possibles défaillances de communication. Le gestionnaire des objets déconnectés, le gestionnaire de connectivité, la fabrique des gestionnaires de connectivité et le gestionnaire du journal, dénotés respectivement *DOM*, *FCM*, *CM* et *LM*, sont dans la même entité d'exécution que *DO* car le cycle de vie de *DO* est géré par *FCM*, lui même géré par *DOM* qui s'occupe aussi de *LM*. Cette deuxième entité d'exécution doit être lancée avant *Client*. À l'exception de *CM*, les gestionnaires sont uniques sur un même terminal mobile.

Soit le client est une application légataire à laquelle sont associés des objets déconnectés via l'intercepteur portable, soit il obtient la référence de Domint par l'intermédiare de *DOM*. *DOM* est le point d'entrée pour trouver les autres gestionnaires afin de les configurer. *CM* est l'entité donnant les informations sur une ressource tel que, si le niveau de disponibilité de cette ressource est insuffisant, le client doit être déconnecté[2]. Des exemples de ressources à contrôler pour la connectivité sont le niveau de la batterie du terminal mobile, le pourcentage de la bande passante du réseau sans fil, et la qualité de la connexion avec un objet ou un hôte distant. Afin de permettre une gestion de la qualité de service différente selon le client, un gestionnaire de connectivité est associé par lien logique entre un client et un objet serveur distant. C'est la granularité la plus fine. Il est facile d'imaginer d'autres politiques telles qu'un gestionnaire de connectivité par application ou par hôte distant. Pour simplifier cette première conception, les objets déconnectés et les gestionnaires de connectivité sont créés sur demande des intercepteurs portables et restent actifs toute la durée de vie de l'intercepteur portable, donc du client. La question du choix de la granularité fait l'objet de travaux en cours [13].

2.2.2 Première requête en connectivité forte

La figure 1-a montre les interactions durant l'envoi de la première requête à l'objet distant dans le cas d'une connectivité forte, c.-à-d. en mode « connecté ». Dans le diagramme, les requêtes 1 et 3 sont des cas particuliers : la requête 1 est interceptée, traitée et retranmise par l'ORB en tant que requête 3. Entre les deux, les actions suivantes sont effectuées.

[1] Dans le papier, une entité d'exécution est un espace d'adressage avec plusieurs fils d'exécution et une seule instance d'ORB. L'intercepteur portable attaché à un client ou à un objet CORBA appartient à la même entité d'exécution que le client ou l'objet qu'il contrôle.

[2] La gestion de ressource est donc interprétée pour la gestion de connectivité.

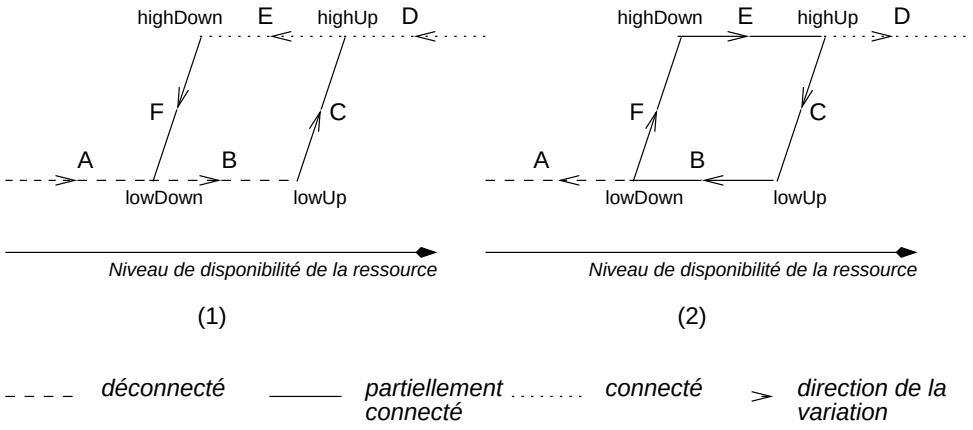
PI tient à jour une table d'associations liant une référence CORBA (IOR, « *Interoperable Object Reference* ») et un gestionnaire de connectivité. La requête étant la première vers cet objet distant, *PI* obtient la référence de *RM* (2.1), s'il ne l'avait pas déjà, et obtient la référence de *CM* auprès de *RM* (2.2). *RM* crée l'objet déconnecté *DO* (2.2.1) et le gestionnaire de connectivité *CM* pour cette nouvelle connexion logique (2.2.2). Durant son initialisation, *DO* obtient les références de *RM* et *LM* (2.2.1.1 et 2.2.1.2), et fournit à *LM* l'objet par valeur CORBA pour l'interprétation des futures requêtes journalisées (2.2.1.3) : cet objet est utilisé dans la figure 1-b. À la fin de cette procédure, *PI* interroge *CM* et décide vers qui la requête du client doit être transmise (*cf*. section 4 pour la table de décision). Dans cette séquence pendant laquelle la connectivité est bonne, *PI* laisse la requête passée vers l'objet distant (3). Par conséquent, dans le mode connecté, comme dans Coda [14], le client accède directement à l'objet distant et l'objet déconnecté *DO* n'est pas au courant de la communication et donc pas à jour. C'est ainsi que *DO*, après s'être assuré que la connectivité le permet (4.1.1 et 4.1.2), obtient une copie (incrémentale) de l'état de l'objet distant qu'il représente (4.2). Bien sûr, les prochaines requêtes en mode connecté ne génèreront pas les requêtes d'initialisation écrites en italique dans la figure 1-a.

2.2.3 Requête en connectivité faible ou nulle

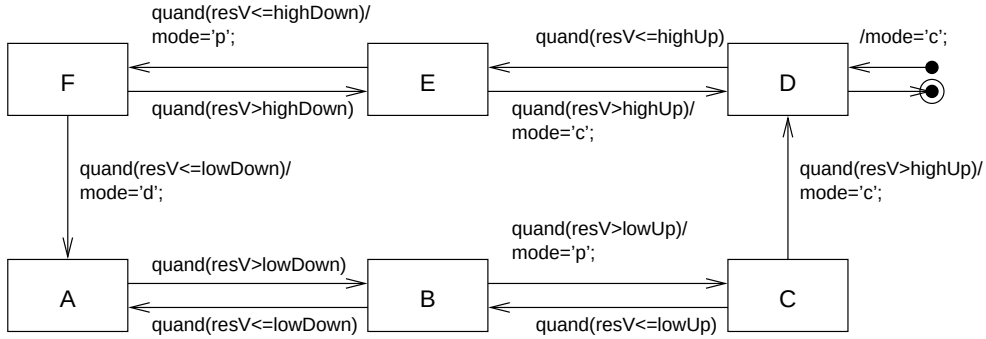
Lorsque la connectivité devient faible ou nulle, *PI* indique à l'ORB de transmettre la requête à *DO* (*cf.* figure 1-b). Les requêtes qui suivent dépendent de l'application puisque *DO* est construit et non pas généré automatiquement à partir de l'objet distant. Nous donnons maintenant deux séquences possibles et souhaitables : 5.a and 5.b (*cf*. section 5 pour des détails sur la conception de *DO* et *LM*).

Si la requête du client est interprétée par *DO* comme fournissant de l'information à l'objet distant, *DO* met à jour son état et prépare une nouvelle requête, appelée une « requête DO ». Le cas le plus simple que nous prenons ici est que ces deux requêtes possèdent le même prototype d'opération écrite en IDL CORBA. *DO* encode sa requête dans un conteneur (un CORBA `Any`) qu'il transmet à *LM* (5.a.1). *LM* essaie (périodiquement) de transmettre la requête (5.a.2.2) après avoir testé la connectivité (non nulle) (5.a.2.1). L'objet par valeur transmis à la requête 2.3 de la figure 1-a sert à décoder la requête avant l'émission. Ainsi, *LM* gère un journal global pour toutes les applications.

La séquence 5.b est choisie lorsque la requête du client retourne de l'information en provenance de l'objet distant. Si le journal est vide (5.b.1) et que la connectivité n'est pas nulle (5.b.2), *DO* envoie sa requête DO (5.b.3) et, au retour de cette dernière, met à jour son état avant de transmettre la réponse au client. Sinon, cas d'une déconnexion, il fournit tout de suite au client une réponse tenant compte de son état (non synchronisé avec l'objet distant).



(a)



"resV" signifie "niveau de disponibilité de la ressource
"c", "p" et "d" signifient "connecté", "partiellement connecté" et "déconnecté", respectivement

(b)

FIG. 2 – L'hystérésis de la gestion de connectivité.

3 Détection de connectivité

Le rôle du gestionnaire de connectivité est de détecter les déconnexions et de construire les trois modes de connectivité : « connecté », « partiellement connecté » et « déconnecté ». La section 3.1 montre que l'introduction du mode partiellement connecté entre les deux autres minimise le nombre de déconnexions et reconnexions, et les prépare. Bien sûr, l'hystérésis construit est paramétrable (*cf.* section 3.2) et permet une déconnexion volontaire par le client (*cf.* section 4).

3.1 Hystérésis pour l'introduction du mode « partiellement connecté »

Les gestionnaires de connectivité utilisent un mécanisme d'hystérésis pour lisser les variations de niveau de disponibilité d'une ressource (*cf.* Figure 2-a). Il définit trois

modes : « déconnecté » quand les requêtes ne sont traitées que par l'objet déconnecté ; « connecté » par l'objet distant seulement ; « partiellement connecté » par les deux. Lors d'une déconnexion volontaire, un transfert d'état s'opère de l'objet distant vers l'objet déconnecté. Réciproquement, la reconnexion implique un transfert du journal de requêtes DO et une réconciliation. L'effet « ping-pong » intervient si le niveau de disponibilité de la ressource oscille autour d'une valeur frontière entre deux modes de connectivité, provoquant des transferts d'états ou de journaux répétés. Dans [9], Harbus note un effet similaire qu'il appelle « effet boomerang » dans le contexte de la migration de processus. La solution classique à ce problème est d'introduire un double seuil. Dans notre cas, puisque l'effet intervient dans deux situations (déconnexion et reconnexion), un mécanisme d'hystérésis comprenant deux doubles seuils est nécessaire. Dans la figure 2-a.1, tant que le niveau de disponibilité croît et reste inférieur à `lowUp` (*resp.* `highUp`), le terminal mobile est déconnecté (*resp.* partiellement connecté). Lorsque le niveau de disponibilité décroît et est supérieur à `highDown` (*resp.* `lowDown`), le terminal mobile est connecté (*resp.* partiellement connecté). Sans la figure 2-a.2, il existe toujours un risque d'effet ping-pong autour des valeurs `highDown` et `lowUp`. Donc, quand le niveau de disponibilité arrive dans l'état `F` à partir de l'état `E` (*resp.* de l'état `B` à partir de l'état `C`) et que le niveau de disponibilité redevient supérieur à `highDown` (*resp.* inférieur à `lowUp`), le mode reste partiellement connecté jusqu'à la valeur `highUp` (*resp.* `lowDown`).

Le diagramme de transition d'états correspondant est dessiné dans la figure 2-b. L'état qui suit immédiatement l'état initial et qui précède immédiatement l'état final est l'état `D`. Ainsi, il est supposé que le client démarre et arrête l'application sur le terminal mobile alors qu'il est fortement connecté. L'hystérésis change d'état même si le client s'est volontairement déconnecté. Par contre, pour décider si *LM* et *DO* peuvent transmettre les requêtes DO, la variable booléenne `forward` dépend du fait d'être volontairement déconnecté : $forward = \neg(mode = d) \ \wedge \ \neg voluntary$, `voluntary` étant égal vrai lors de déconnexions volontaires.

3.2 Paramétrage de l'hystérésis

L'hystérésis permet de faire face à des applications très différentes dans des environnements variés. Les valeurs des différents seuils (`lowDown`, `lowUp`, `highUp` et `highDown`) doivent être choisies avec précaution. En outre, elles dépendent du type de ressource. Prenons comme exemple la bande passante disponible du réseau sans fil, les valeurs représentant des pourcentages. Des seuils bas (positionnant l'hystérésis vers la gauche) correspond à une vision pessimiste dans laquelle l'utilisateur s'attend à une faible bande passante et fixe les valeurs pour que le terminal mobile reste connecté même pour une disponibilité de la bande passante très basse. Au contraire, des valeurs hautes pour les seuils (déplaçant l'hystérésis vers la droite) représente une vision optimiste dans laquelle le client considère que le réseau sans fil est très performant et se déconnecte dès que le réseau faiblit un peu. Il est aussi possible de configurer un hystérésis très large soit en fixant les seuils `lowDown` et `lowUp` à des valeurs faibles, et `highDown` et `highUp` à des valeurs élevées, soit en fixant `lowDown` et `highDown` à des valeurs faibles, et `lowUp` et `highUp` à des valeurs élevées, privilégiant ainsi le mode partiellement connecté pour un envi-

ronnement très instable avec des variations importantes fréquentes. En guise de dernier cas, choisir une configuration avec un hystérésis étroit (valeurs des seuils rapprochées les unes des autres) est déconseillé, car l'hystérésis est alors traversé très rapidement, au risque d'observer l'effet ping-pong. En outre, l'agilité de Domint et sa capacité à s'adapter aux changements brusques de l'environnement est fonction de la fréquence d'échantillonage de la disponibilité de la ressource contrôlées.

Enfin, l'hystérésis est reconfigurable, signifiant par la-même que les valeurs des seuils peuvent être modifiées pendant l'exécution ou sur décision de l'utilisateur via une interface graphique construite à cette effet. Domint peut alors s'adapter à des conditions opératoires différentes telles que les déconnexions courtes lors de changements de cellules dans un réseau sans fil et les déconnexions longues lors de voyages en avion [6].

4 Commutation transparente entre l'objet distant et l'objet déconnecté

Le mécanisme des intercepteurs portables CORBA [16] est conçu pour permettre l'ajout de services extra-fonctionnels aux objets d'une application. Nous l'utilisons dans Domint du côté du client sur le terminal mobile pour effectuer la commutation transparente entre l'envoi des requêtes directement vers l'objet distant et le déroutement vers l'objet déconnecté correspondant. Pour opérer la commutation, un intercepteur d'IOR est inséré du côté du serveur pour ajouter un composant aux profils des références lors de la création des objets. Ce composant contient un booléen spécifiant si l'objet serveur doit posséder un mandataire sur le terminal mobile. Ainsi, lors de l'envoi d'une requête, si la référence de l'objet cible contient le nouveau composant et que la valeur du booléen est « vrai », l'intercepteur du côté client effectue le traitement suivant sur la requête et la réponse. Selon le changement d'état dans l'hystérésis, l'intercepteur peut construire une exception CORBA de type `ForwardRequest` qui indique à l'ORB que la requête arrêtée par l'ORB et transmise à l'intercepteur, doit être ré-émise avec une autre référence cible, celle de l'objet déconnecté, et *vice-versa*.

Le tableau 1 présente la table de décision contenue dans *PI* pour la commutation transparente. Les événements déclencheurs sont les requêtes du client. Les deux premiers paramètres en entrée sont les variables booléennes `voluntary` et `direct`, indiquant respectivement si le client a demandé une déconnexion volontaire et si la requête précédente a été émise directement vers l'objet distant (sans passer vers l'objet déconnecté). Les deux paramètres suivants sont le mode et le nom de l'opération appelée sur l'objet distant (`opName` dans le tableau). Les opérations effectuées occupent les quatre colonnes de droite : inverser les variables booléennes `voluntary` et `direct`, et lever l'exception `ForwardRequest` (`ForwardExcep` dans le tableau) en changeant l'objet cible pour commuter soit vers l'objet déconnecté, soit vers l'objet distant (`RO` pour « *Remote Object* » dans le tableau). Lever l'exception `ForwardRequest` implique la ré-émission de la même requête vers un autre objet cible, cette deuxième requête étant elle-aussi interceptée mais traitée différemment par *PI* puisque les variables booléennes `voluntary` ou `direct` ont changé de valeur. Enfin, comme indiqué dans le tableau 1, le client se déconnecte et se récon-

voluntary	direct	mode	opName	inverser voluntary	inverser direct	lever ForwardExcep (DO)	lever ForwardExcep (RO)
vrai	faux	d	disconnect				
			reconnect	+			
			autre				
		p	disconnect				
			reconnect	+			
			autre				
		c	disconnect				
			reconnect	+			
			autre				
faux	vrai	d	disconnect	+	+	+	
			reconnect				
			autre		+	+	
		p	disconnect	+	+	+	
			reconnect				
			autre		+	+	
		c	disconnect	+	+	+	
			reconnect				
			autre				
	faux	d	disconnect	+			
			reconnect				
			autre				
		p	disconnect	+			
			reconnect				
			autre				
		c	disconnect	+			
			reconnect				
			autre		+		+

TAB. 1 – La table de décision de la commutation dans *PI*.

necte volontairement en appelant respectivement les opérations `disconnect` et `reconnect` sur l'objet distant. Les deux opérations atteignent l'objet déconnecté : la commutation est effectuée avant l'exécution de `disconnect` mais après `reconnect` (la commutation est effective uniquement lors de la requête qui suit, si le mode la permet toujours). La table de décision autorise n'importe quelle séquence d'opérations `disconnect` et `reconnect`.

5 Conception des objets déconnectés et journalisation

La conception des objets déconnectés est dépendante de l'application. Cette section présente quelques patrons de conception ainsi que des questions ouvertes sur la conception de ces objets pour les deux modes pendant lesquels ils sont traversés par les requêtes du client (*cf.* sections 5.1 et 5.2). La section 5.3 détaille le mécanisme de

journalisation.

5.1 Mode partiellement connecté

Dans le mode partiellement connecté, les opérations sont exécutées par l'objet déconnecté et l'objet distant dans un ordre dépendant de la sémantique de l'opération. Dans cette première version de la plate-forme, les opérations possédant uniquement des paramètres en entrée (mot clé « `in` » en IDL CORBA) sont exécutées d'abord localement, l'objet distant est mis à jour ensuite. Si le prototype de l'opération ne contient que des paramètres en sortie (« `out` ») ou une valeur de retour, l'opération est d'abord exécutée par l'objet distant, puis l'objet déconnecté mis à jour. Domint ne permet pas actuellement le mixage de paramètres « `in` » et « `out` », ni les paramètres « `inout` ». Une autre question ouverte est la gestion des exceptions renvoyées par l'objet distant.

5.2 Mode déconnecté

Dans le mode déconnecté, les opérations ne sont exécutées que par l'objet déconnecté. Si son prototype ne contient que des paramètres en entrée, l'opération est journalisée. Dans le cas de paramètres en sortie et d'une valeur de retour, la requête est journalisée si elle provoque un changement d'état devant être transmis à l'objet distant. Par exemple, pour un lecteur de courriers électroniques, il est intéressant de connaître les messages lus par le client [4]. Lors de la reconnexion, le journal d'opérations est transmis et la réconciliation est atomique ; la réconciliation progressive telle que réalisée dans Coda pour des fichiers [25], est une question ouverte. Clairement, l'exécution pendant les déconnexions n'est pas « équivalente » (en terme de séquence d'opérations) à l'exécution sans déconnexions. Cela est acceptable pourvu que l'utilisateur ait connaissance du mode de connectivité via un icône ou une interface graphique.

5.3 Journalisation des opérations

L'interprétation des requêtes journalisées est dépendante de l'application. Le code qui analyse et retransmet les requêtes est fourni par l'objet déconnecté lors de son initialisation via un objet par valeur CORBA (*cf.* requête 2.2.1.3 de la figure 1-a). En plus d'opérations pour l'analyse et la retransmission de requêtes journalisées, l'interface abstraite de ces objets par valeur pourra inclure des opérations de traitement du journal pour optimisation, par exemple le compactage [14].

Les techniques de recouvrement par points de reprise sont justifiées dans le cas de déconnexions volontaires mais leur utilisation pour gérer des déconnexions inopinées reste un problème ouvert [2]. Les fautes intermittentes survenant fréquemment dans les réseaux sans fil, elles nécessitent des mécanismes de détection de fautes différents de ce qui est utilisé dans les réseaux traditionnels. De plus, la constitution des points de reprise elle-même doit prendre en compte les deux modes de déconnexion.

Comme vu précédemment, afin de faciliter les réconciliations, nous avons choisi de journaliser les opérations et non les changements d'état. Dans le prototype existant, nous considérons que les objets serveurs présents sur le réseau fixe ne sont pas accédés de façon concomitante, y compris pendant la déconnexion. Ainsi la réconciliation est simplifiée et la transition du mode déconnecté vers le mode partiellement connecté nécessite uniquement une réexécution des opérations journalisées par la copie locale. Des travaux sont en cours pour permettre à la fois des accès concurrents aux données locales et distantes [7], et pour concevoir des algorithmes de réconciliation adaptés au maintien de la cohérence de données en environnements mobiles et basés sur les transformées opérationnelles [30].

6 Résultats de performance

Nous avons mené une série de tests de performance sur différentes combinaisons de logiciel et matériel : Ordinateur PC portable utilisant GNU/Linux RedHat 7.2 ou Microsoft Windows2000, et un assistant personnel numérique iPAQ (206 Mhz, 16 MB de ROM et 32 MB de RAM) utilisant GNU/Linux Familiar 0.6 ou Microsoft WindowsCE. Les communications sans fil sont assurées par une carte Compaq IEEE 802.11b WL110 à 11Mbps dans chaque terminal et une station de base logicielle Compaq IEEE 802.11b WL100 + WL300. Chaque test a été exécuté 1000 fois afin d'avoir des moyennes significatives et des intervalles de confiance avec une probabilité de 99%. Tous les programmes étant écrits en Java, un appel explicite aux ramasseurs de miettes est effectué avant chaque exécution côtés client et serveur. Nous avons utilisé la machine virtuelle IBM J9 v1.2.2[3]. Des mesures ont également été réalisées avec la machine virtuelle Classic Blackdown-1.3.1-RC1 sur Familiar, et celle-ci s'est révélée moins performante pour des requêtes de petite taille (jusqu'à 70% de moins) mais beaucoup plus rapide pour des messages de moyenne et grande taille (jusqu'à 270%).

Nous présentons tout d'abord une évaluation de mesures de base indépendemment de Domint. Nous analysons ensuite les résultats des performances de Domint et identifions les coûts de ses différentes composantes par une analyse par régression linéaire.

6.1 Mesures de base

Nous avons mesuré les performances de base pour le traitement de requêtes simples[4] via des sockets TCP, et en utilisant ORBacus 4.1.0 sans intercepteurs puis avec des intercepteurs n'effectuant aucun traitement (*cf.* table 2). Pour toutes ces expérimentations, le logiciel Domint n'est pas impliqué.

Comme attendu, les traitements locaux sont en général plus rapides que les traitements à distance, sauf sur WindowsCE dans le cas de requêtes de petite taille (de 1B à 128B). Une comparaison des résultats pour WindowsCE et Familiar montre des différences selon la taille des requêtes et en fonction de l'utilisation d'un ORB ou non. Pour

[3] Aucune difficulté d'utilisation n'a été rencontrée sur les différentes combinaisons de matériel et logiciel.

[4] Pour plus de clarté, seuls les résultats avec des paramètres '`out`' (en sortie) pour iPAQ sont indiqués.

iPAQ - MS WindowsCE

Expérience	1 B	128 B	16 KB	128 KB	512 KB	1 MB
1. TCP–L	7.0 ±0.0	7.1 ±0.0	35.6 ±0.1	243.6 ±0.3	951.8 ±0.6	1904.0 ±0.5
2. TCP–R	3.7 ±0.1	4.2 ±0.1	54.1 ±3.1	446.5 ±10.0	1751.1 ±17.9	3487.5 ±23.2
3. WOPI–L	17.9 ±0.2	17.8 ±0.2	48.6 ±0.2	296.8 ±0.3	1241.3 ±0.4	2306.3 ±0.5
4. WOPI–R	12.1 ±0.1	13.8 ±2.0	65.4 ±3.4	456.2 ±6.6	1870.0 ±17.1	3707.3 ±21.6
5. PI–L	21.6 ±0.2	21.9 ±0.2	53.3 ±0.2	300.9 ±0.3	1270.0 ±0.4	2341.6 ±0.5
6. PI–R	16.3 ±1.9	17.4 ± 1.9	68.1 ±2.9	462.8 ±7.3	1898.7 ±11.6	3707.8 ±14.9

iPAQ - GNU/Linux Familiar

Expérience	1 B	128 B	16 KB	128 KB	512 KB	1 MB
7. TCP–L	0.9 ±0.0	1.0 ±0.1	3.7 ±0.1	28.2 ±0.7	105.7 ±1.7	223.2 ±1.7
8. TCP–R	4.6 ±0.0	5.1 ±0.0	62.3 ±3.3	436.7 ±5.7	1737.5 ±5.7	3494.6 ±23.2
9. WOPI–L	21.5 ±0.6	20.8 ±0.5	29.6 ±0.5	83.9 ±0.2	269.4 ±0.2	531.4 ±0.2
10. WOPI–R	49.5 ±0.7	49.5 ±0.6	68.9 ±6.4	461.7 ±3.7	1810.1 ±5.2	3702.5 ±7.5
11. PI–L	24.5 ±0.5	23.1 ±0.4	32.5 ±0.4	87.8 ±0.2	282.1 ±0.2	554.8 ±0.5
12. PI–R	53.1 ±0.9	56.1 ±0.6	69.4 ±3.2	464.2 ±3.7	1818.7 ±5.4	3712.3 ±7.5

TAB. 2 – Les durées de base (ms) pour différentes tailles de requêtes (octets) : '*TCP*', '*WOPI*' et '*PI*' signifient 'sockets TCP', 'ORB sans intercepteur portable côté client', 'ORB avec intercepteur portable côté client vide', respectivement ; '*L*' et '*R*' signifient 'client et serveur colocalisés sur l'iPAQ', 'serveur sur l'hôte distant sur le réseau fixe', respectivement.

des requêtes TCP locales, Familiar est significativement meilleur (de 80% à 90%). Avec un ORB, WindowsCE donne des temps de réponse meilleurs pour les messages de 1B à 128B tandis que Familiar est meilleur pour les requêtes de grande taille, avec une différence de 40% à 75%. Le coût de l'ORB peut être déduit en comparant les temps obtenus avec TCP et avec un ORB mais sans intercepteurs. L'impact de l'ORB est moindre avec Familiar.

Une autre mesure intéressante concerne le coût des intercepteurs portables. En comparant les résultats sans intercepteurs (lignes 3–4 et 9–10) et avec intercepteurs (lignes 5–6 et 11–12), le surcoût va de 7% à 13% sur Familiar et de 1% à 33% sur Windows, avec le maximum pour les messages de petite taille. Bien que non négligeable, le coût des intercepteurs apparaît comme raisonnable en regard de la taille des messages. Le surcoût que nous avons mis en évidence est plus faible que dans [31], où les auteurs ont mené des expérimentations avec deux stations de travail UltraSPARC-2 à double processeur utilisant SunOS 5.7 et ont obtenu une pénalité de 26% en utilisant des intercepteurs vides.

6.2 Mesures avec Domint

Dans cette section, nous analysons les résultats des performances de Domint. Notre objectif est d'identifier précisément le coût de chacune des composantes de Domint. Nous ne détaillons donc pas toutes les mesures effectuées mais présentons les équations uti-

lisées pour une analyse par régression linéaire et donnons les conclusions de cette analyse.

En se basant sur les diagrammes de collaboration décrits dans la figure 1, les résultats de performance de Domint présentés dans la figure peuvent être déduits des chiffres donnés dans la table 2 en utilisant les équations suivantes[5], « conn », « part » et « disc » signifiant connecté, partiellement connecté et déconnecté, et « `O` » et « `I` » indiquant si les arguments sont en sortie ou en entrée :

$$t_{conn-O} = t_{conn-I} = t_{PI-L-short} + t_{PI-R} + t_{conn\_mngt}$$
$$t_{part-O} = t_{conn-O} + t_{PI-L} + 2 * t_{request\_intra\_VM}$$
$$t_{part-I} = t_{part-O} + t_{logging}$$
$$t_{disc-O} = t_{part-O} - t_{PI}$$
$$t_{disc-I} = t_{part-I} - t_{PI}$$

avec :

- $t_{PI-L-short}$: durée d'envoi d'une requête locale de petite taille (entre 1B et 128B) via l'ORB,
- t_{conn_mngt} : durée de calcul liée à l'hystérésis au niveau du gestionnaire de connectivité,
- $t_{request\_intra\_VM}$: durée de traitement d'une requête entre deux objets CORBA colocalisés dans la même unité d'exécution (machine virtuelle),
- $t_{logging}$: durée d'empaquetage d'une requête sous la forme d'un `Any` CORBA et d'enregistrement dans le journal.

Pour WindowsCE et Familiar, l'analyse par régression linéaire fournit de très bons coefficients de corrélation et de détermination (supérieurs à 0.99) en ce qui concerne les performances sur TCP. Les coefficients restent bons pour les performances avec un ORB pour WindowsCE, mais diminuent pour Familiar (autour de 0.6 et 0.4 respectivement). Cependant, une comparaison des chiffres de performance obtenus au moyen des équations et ceux donnés par mesure donne une différence plus importante pour WindowsCE que pour Familiar.

Nous analysons à présent le surcoût lié à la gestion de la connectivité et à la journalisation. Nous donnons en premier le pourcentage correspondant aux messages de petite taille et ensuite celui obtenu pour des messages de grande taille. Nous avons logiquement observé dans tous les cas un surcoût plus important pour les messages de petite taille. Pour WindowsCE, les surcoûts correspondant à conn-`O`, part-`O`, disc-`O` varient de 70% à 2%, de 52% à 21%, et de 77% à 5%, respectivement. Pour Familiar, les surcoûts varient de 14% à 1%, de 50% à 6%, et de 20% à 6%. Ceci suggère que WindowsCE est plus sensible à la présence d'un processus « *multithreadé* » comprenant les objets de Domint suivants : *DOM*, *RM*, *CM*, et *LM* (cf. figure 1). Ceci implique également que le coût de la journalisation n'est pas négligeable.

Par ailleurs, le surcoût lié directement à l'utilisation de l'intercepteur portable *PI* peut être calculé à partir des mesures de base. Pour WindowsCE, les valeurs de conn–`O`, part–`O` et disc–`O` s'échelonnent de 71% à ≈0%, de 80% à 40%, et de 77% à 5%, respec-

[5]Le tableau des mesures avec Domint est disponible dans [5].

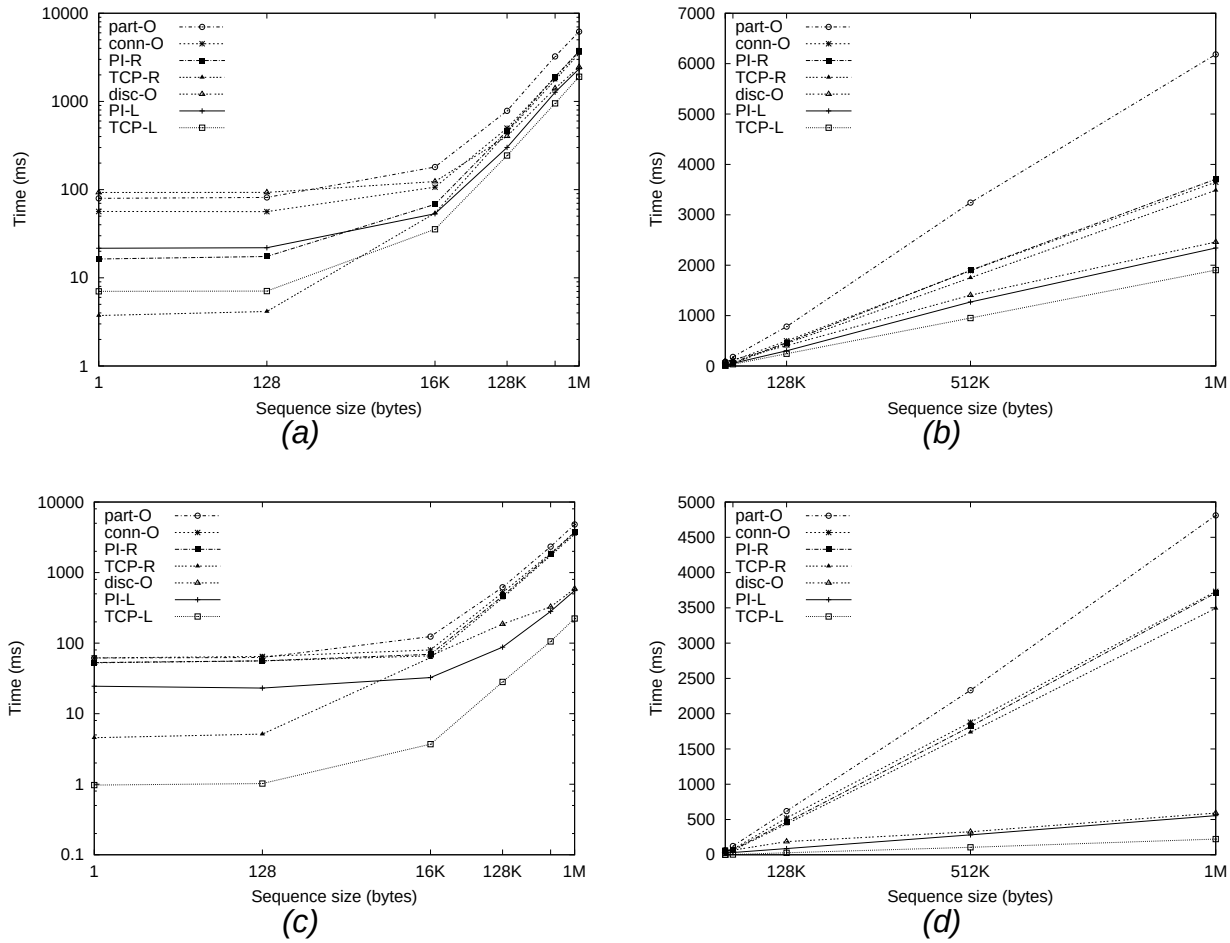


FIG. 3 – Mesures de base et avec Domint : *(a)* et *(b)* iPAQ exécutant Microsoft WindowsCE, *(c)* et *(d)* exécutant GNU/Linux Familiar.

tivement. Pour Familiar, ces mêmes indicateurs varient de 14% à 1%, de 59% à 6%, et de 40% à 13%, respectivement.

7 Travaux connexes

Un certain nombre de projets de recherche traitent de l'adaptation des applications pour l'accès aux informations dans les réseaux sans fil. Dans cette section, nous distinguons les travaux séminaux non basés sur des intergiciels de ceux plus récents mettant en œuvre des services avancés tels que ceux fournis par CORBA. Pour une synthèse sur l'adaptation des approches clients/serveurs aux environnements sans fil, le lecteur peut se référer à [10].

Coda est la première contribution conséquente proposant le concept d'opérations déconnectées sur des fichiers, et fonctionnant de façon transparente en connectivité faible

ou déconnexion [12, 14, 27]. Successeur de Coda, Odyssey démontre l'utilité de la collaboration entre l'application et le système afin d'améliorer l'agilité et la fidélité [15]. Coda et Odyssey sont dédiés à la manipulation de fichiers ; ce modèle de bas niveau (comparé au modèle objets de CORBA) gère des entités à gros grains (des fichiers). Nous pensons que les objets introduisent une flexibilité plus importante. Coda et Odyssey sont maintenant inclus dans le projet Aura abordant l'informatique ubiquiste, et plus spécifiquement le problème de la distraction de l'attention de l'utilisateur, qui est selon les auteurs la ressource la plus précieuse [26]. Aura manipule des tâches, capturant l'intention de l'utilisateur [28]. Par contraste avec les scénarios présentés dans [28] dans lesquels l'infrastructure permet aux utilisateurs de faire migrer leurs tâches d'un environnement à un autre, Domint se concentre sur la continuité de service lors des déconnexions.

Le projet Rover [11] propose une infrastructure pour gérer les variations de disponibilité des ressources et les déconnexions entre un terminal mobile et des serveurs fixes. Les deux concepts clés sont les objets dynamiques relogeables (RDO) et les files de RPC (QRPC). Un RDO est un morceau de code et de données qui peut être chargé (copié) et déchargé ; le modèle objet est spécifique et ne s'appuie pas sur un modèle « standard ». C'est l'application qui contrôle le placement des RDO selon la disponibilité des ressources (bande passante et puissance de calcul). Rover gère deux types d'ensembles de ces objets : un ensemble par application et un ensemble global au terminal mobile. Dans Domint, les objets déconnectés étant des objets CORBA, ils sont de fait globaux. Le mécanisme des QRPC gère les interactions entre le terminal mobile et les serveurs fixes de manière transparente : les RPC sont empilés et transmis uniquement lorsque le réseau sans fil le permet. Cette architecture est très générique et flexible. Cependant, les programmeurs doivent suivre le modèle objet défini et programmer les mouvements d'objets. Dans Domint, la tâche du programmeur est simplifiée dans le sens où le modèle objet est standard (support des applications légataires CORBA), les changements dans l'application étant concentrés dans la conception des objets déconnectés.

Dans le contexte CORBA, le service de messagerie de l'OMG [18] fournit des squelettes offrant des invocations dites « asynchrones » ou « dépendant du temps » qui utilisent des extensions du protocole GIOP. Ces extensions gèrent la mise en file d'attente et la re-transmission des requêtes. Le premier mécanisme permet au client de soumettre une requête sans se bloquer en attente de la réponse. Ultérieurement, le client reçoit la réponse soit par un appel ascendant (« *callback* ») venant de l'ORB, soit par une scrutation (« *polling* »). Avec le deuxième mécanisme, le client émet une requête, se déconnecte, puis se reconnecte plus tard pour prendre la réponse. La spécification présente une architecture typique avec des routeurs : des routeurs du côté du client et du serveur, et d'autres routeurs intermédiaires. Cette architecture est similaire aux QRPC de Rover. Avec Domint, nous soutenons l'idée qu'à partir du moment où le client visualise les informations de connectivité, s'il continue à travailler alors qu'il est déconnecté, c'est qu'il désire une réponse non différée à ses requêtes (continuité de service), même si elles diffèrent des réponses hypothétiques en mode connecté. Par conséquent, seul est journalisé sur le terminal mobile ce qui est nécessaire pour la réconciliation : toutes les requêtes n'ont pas besoin d'être journalisées et le journal peut être optimisé (compactage. . .). Un autre mécanisme de méta-programmation CORBA, dual du mécanisme des

intercepteurs portables, est susceptible d'être utilisé sur le terminal mobile : les « mandataires intelligents » (« *smart proxies* ») [31]. Il s'agit de modifier la souche du côté client pour y inclure le code mis dans les intercepteurs portables. Ce mécanisme est plus performant et moins intrusif (n'intervenant que pour les objets serveurs possédant effectivement des objets déconnectés). En contre-partie, nous perdons la portabilité (au niveau ORB).

Toujours dans le monde CORBA, Π^2 [23, 24] et ALICE [8, 1] traitent du problème du changement de cellules dans les réseaux sans fil lorsque l'utilisateur est mobile, donc des déconnexions de courtes durées. Π^2 introduit deux objets mandataires (un sur le terminal mobile, l'autre sur un nœud du réseau fixe) ; les déconnexions sont ainsi transparentes. ALICE utilise aussi un mandataire sur un nœud du réseau fixe pour les déconnexions volontaires et involontaires. Domint ne requiert aucun mandataire sur le réseau fixe et le niveau auquel les déconnexions sont gérées dans le client est différent (gestion transparente dans un intercepteur portable pour Domint et gestion non transparente dans le client lors du retour d'une exception par l'ORB pour Alice).

8 Conclusion et perspectives

Dans cet article, nous avons montré comment la technologie actuelle des intergiciels permet à des applications légataires CORBA de fonctionner en environnement mobile sans modification aucune du côté client (sur le terminal mobile) et avec seulement quelques modifications mineures du côté serveur[6]. Bien que le prototype réalisé utilise CORBA, les concepts mis en oeuvre sont suffisamment généraux pour être appliqués à d'autres types d'intergiciels, dans la mesure où ils fournissent des mécanismes d'interception et de transfert d'objets par valeur.

Un service de configuration peut être conçu pour indiquer en permanence l'état de la connectivité via une interface graphique appropriée et offrir à l'application le moyen de provoquer une déconnexion volontaire. Les déconnexions involontaires étant gérées entièrement par l'intergiciel, aucune modification n'intervient dans le code applicatif. Comme il est aujourd'hui possible d'embarquer un intergiciel sur étagère complet respectant la version 2.4 de CORBA sur un assistant personnel numérique de type iPAQ, il est possible de s'appuyer sur le mécanisme des intercepteurs portables pour implanter des politiques d'adaptation transparente pour l'application.

La plate-forme Domint est ouverte et extensible, et permet une adaptation spécifique à l'application. Par exemple, le gestionnaire de journaux peut bénéficier d'objets par valeur propres à l'application pour déterminer dynamiquement quelles informations doivent être enregistrées. D'autre part, nous avons proposé un mécanisme d'hystérésis permettant d'éviter des changements d'état continuels lorsque le niveau de connectivité subit des oscillations autour d'une valeur de changement de mode de connectivité.

Nous avons présenté les performances d'un prototype CORBA réalisées sur un ordinateur portable PC et un assistant numérique iPAQ. Les performances obtenues montrent

[6] La seule modification est l'affectation, POA par POA, de la politique « déconnectable » pour indiquer si les objets serveur doivent posséder un mandataire (objet déconnecté) sur l'hôte client.

que le surcoût introduit par la plate-forme Domint est acceptable pour un utilisateur final.

Un de nos objectifs est maintenant de compléter Domint en ajoutant la gestion de la cohérence des objets manipulés, ce qui implique la mise en oeuvre d'algorithmes de réconciliation adaptés au contexte de la mobilité [7]. Nous avons choisi de bénéficier des recherches en travail collaboratif basées sur les transformées opérationnelles [30]. Une deuxième perspective concerne la prise en compte de plusieurs modes de communication. Domint ne gère que les invocations synchrones. Or, les communications asynchrones sont naturellement pertinentes en environnement mobile et sont de plus en plus utilisées dans les intergiciels via des services d'événements ou de notification. Ce point est actuellement étudié dans le cadre de la conception d'une plate-forme à base de composants à partir de Domint s'appuyant sur l'implantation OpenCCM [21] des composants CORBA [20]. Un dernier axe de travail est d'étudier les mécanismes de réflexion, autres que les intercepteurs, pour formaliser et compléter la collaboration entre l'application et l'intergiciel en environnement mobile [3]. Nous nous intéressons à la définition de métadonnées permettant de préciser les objets déconnectables, puis ceux nécessaires sur le terminal mobile pendant la déconnexion, et enfin leur priorité en cas de restriction mémoire [13].

Références

[1] G. Biegel, V. Cahill, and M. Haahr. A Dynamic Proxy Based Architecture to Support Distributed Java Objects in a Mobile Environment . In *Proc. Int. Symposium on Distributed Objects and Applications (DOA'02)*, pages 809–826, Oct. 2002.

[2] G. Cao and M. Singhal. Mutable Checkpoints : A New Checkpointing Approach for Mobile Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(2) :157–172, February 2001.

[3] L. Capra, G. Blair, C. Mascolo, W. Emmerich, and P. Grace. Exploiting Reflection in Mobile Computing Middleware. *ACM Mobile Computing and Communications Review*, 6(4) :34–44, October 2002.

[4] D. Conan, S. Chabridon, and G. Bernard. Disconnected Operations in Mobile Environments. Ft. Lauderdale, Florida (USA), April 2002.

[5] D. Conan, S. Chabridon, O. Villin, and G. Bernard. Domint : Weak connectivity and disconnected corba objects on hand-held devices. Technical report, Institut National des Télécommunications, Évry (France), May 2003.

[6] D. Conan, S. Chabridon, O. Villin, G. Bernard, A. Kotchanov, and T. Saridakis. Handling Network Roaming and Long Disconnections at Middleware Level. In *Workshop on Software Infrastructures for Component-Based Applications on Consumer Devices (in conjonction with EDOC 2002)*, Lausanne, Switzerland, Sept. 2002.

[7] L. Debassen, S. Chabridon, and G. Bernard. Intergiciel pour l'informatique mobile : Réplication optimiste et réconciliation. In *Soumis à MAJECSTIC'03 - Manifestation des JEunes Chercheurs STIC*, 2003.

[8] M. Haahr, R. Cunningham, and V. Cahill. Supporting CORBA Applications in a Mobile Environment. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking (MobiCom'99)*, Seattle, Washington, USA, 1999.

[9] R.S. Harbus. Dynamic process migration : To migrate or not to migrate. Technical report csri-42, University of Toronto, Toronto, Canada, July 1986.

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

[11] A.D. Joseph, J.A. Tauber, and F. Kaashoek. Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers*, 46(3), 1997.

[12] J.J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1), February 1992.

[13] N. Kouici, D. Conan, and G. Bernard. Disconnection Metadata for Distributed Applications in Mobile Environments. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada (USA), June 2003.

[14] L.B. Mummert, M.R. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP'95)*, Copper Mountain resort,CO, December 1995.

[15] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP'97)*, 1997.

[16] OMG. Portable Interceptors. Interceptors Finalization Task Force published draft, Object Management Group, April 2000.

[17] OMG. *Minimum CORBA*, OMG Document formal/01-02-01 Chapter 23. In [18], February 2001.

[18] OMG. The Common Object Request Broker - Architecture and Specifications. Revision 2.4.2. OMG Document formal/01-02-01, Object Management Group, February 2001.

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

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

[21] http ://www.objectweb.org/openccm/, February 2003.

[22] K. Petersen, M.J. Spreitzer, D.B. Terry, M.M. Theimer, and A.J. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proc. 16th ACM Symposium on Operating Systems Principles*, pages 288–301, Saint Malo, France, October 1997.

[23] R. Ruggaber and J. Seitz. Using CORBA Applications in Nomadic Environments. In *Proc. 3d IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'2000)*, Monterey, CA (USA), December 2000.

[24] R. Ruggaber, J. Seitz, and M. Knapp. π^2 - A Generic Proxy Platform for Wireless Access and Mobility. In *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'2000)*, Portland, Oregon, July 2000.

[25] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1) :26–33, February 1996.

[26] M. Satyanarayanan. Pervasive Computing : Vision and Challenges. *IEEE Personal Communications*, August 2001.

[27] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E. Siegel, and D.C. Steere. Coda : A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computer*, 39(4), April 1990.

[28] J.P. Sousa and D. Garlan. Aura : an Architectural Framework for User Mobility in Ubiquitous Computing Environment. In *Proc. 3rd Working IEEE/IFIP Conference on Software Architecture*, Montreal (Canada), August 2002.

[29] D.B. Terry, M.M. Theimer, K. Petersen, and A.J. Demers. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proc. 15th ACM Symposium on Operating Systems Principles*, pages 172–183, December 1995.

[30] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proc. ACM CSCW*, Dec. 2000.

[31] N. Wang, K. Parameswaran, and D. Schmidt. The Design and Performance of Meta-Programming Mechanisms for Object Request Broker Middleware. In *Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'01)*, San Antonio, TX (USA), Jan. 2001.