Kalimucho : Plateforme d'Adaptation des Applications Mobiles

Christine Louberry, Philippe Roose, Marc Dalmau

Kalimucho :
Plateforme d'Adaptation des Applications Mobiles

Christine Louberry
Ecole des Mines de Nantes
ASCOLA, LINA
Nantes, France
christine.louberry@mines-nantes.fr

Philippe Roose, Marc Dalmau
IUT de Bayonne
T2I, LIUPPA
Anglet, France
{philippe.roose;marc.dalmau}@iutbayonne.univ-pau.fr

Résumé— **L'utilisation de plus en plus fréquente des technologies mobiles nous amène à faire face à de nouveaux défis afin de satisfaire les utilisateurs. De la même façon qu'ils utilisent leurs applications favorites sur leur ordinateur, les utilisateurs souhaitent pouvoir également les utiliser sur leur Smartphone ou leur tablette et que les applications prennent compte de leur position, du temps ou de toute autre information contextuelle. Cependant, de tels systèmes sensibles au contexte impliquent de prendre en compte trois principales caractéristiques : la variation du contexte, la mobilité et les ressources limitées des appareils. Dans cet article, nous essayons de traiter ces caractéristiques par l'adaptation dynamique des applications guidée par la qualité de service (QdS). Nous proposons une plateforme de reconfiguration basée service appelée Kalimucho. Elle implémente une heuristique de déploiement contextuel permettant de trouver une configuration satisfaisant les conditions de contexte et de QdS. Kalimucho a été testée avec le modèle de composant Osagaia/Korrontea et plusieurs périphériques ; les résultats confirment que Kalimucho fournit des adaptations en un temps d'exécution satisfaisant.**

Mots-clés- architecture logicielle; gestion de la qualité de service; sensibilité au contexte; déploiement contextuel; adaptation dynamique; composant logiciel

I. INTRODUCTION

L'utilisation de plus en plus fréquente des technologies mobiles nous amène à faire face à de nouveaux défis afin de satisfaire les utilisateurs. Ces derniers souhaitent pouvoir utiliser leurs applications favorites sur n'importe quel périphérique. De plus, ils souhaitent que les applications puissent être automatiquement personnalisées en fonction de leur position, la luminosité ou toute autre information de contexte. Cependant, nous devons faire face aux principales caractéristiques de ces systèmes : la variation du contexte, la mobilité et la limitation des ressources des appareils.

L'objectif de cet article est de répondre aux besoins de l'utilisateur et aux changements de l'environnement par l'adaptation dynamique des applications et de leur déploiement afin de satisfaire les conditions de qualité de service. Nous nous intéressons particulièrement à la gestion de la QdS des applications distribuées face aux ressources restreintes et à la mobilité des appareils, aux besoins de l'utilisateur et aux contraintes d'utilisation.

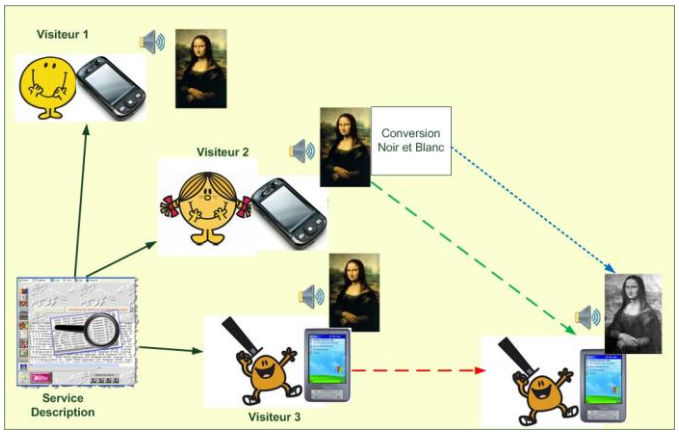

Figure 1. Application de visite d'un musée

Dans cet article, nous illustrons nos travaux par un exemple d'application de visite d'un musée. Le musée propose un service de description des œuvres sous différents médias que les utilisateurs peuvent visualiser sur leur Smartphone (fig. 1). Pour cette application, nos considérons que la meilleure QdS est de fournir une vidéo en couleur. Le paragraphe qui suit décrit un scenario possible d'utilisation de l'application. Lorsque le visiteur 3 se déplace dans le musée, deux cas peuvent se présenter. Premièrement, le débit est faible mais le périphérique est toujours à portée. Afin d'assurer la continuité du service, une solution est de réduire le nombre de données à transmettre. Une solution est d'effectuer un traitement sur une vidéo telle qu'une conversion en noir et blanc ou une réduction de la résolution des images. Deuxièmement, le périphérique n'est plus à portée du serveur, nous devons donc chercher une nouvelle route afin d'atteindre le périphérique 3 et d'assurer la continuité du service. Une solution peut être d'utiliser le périphérique du visiteur 2, qui utilise également le service, afin de lui faire jouer le rôle de relais pour le périphérique 3.

Découvrir une nouvelle route pour transmettre l'information est un problème classique, des protocoles comme AODV [5] peuvent être implémentés. Néanmoins, ces

protocoles ne gèrent que des contraintes de routage physique. Nous utilisons une plateforme de supervision distribuée sur chaque périphérique permettant d'avoir une vue globale de l'application. Ceci nous permet de pouvoir proposer des solutions non uniquement basées sur des critères techniques de faisabilité. En effet puisque les utilisateurs 1 et 2 reçoivent la vidéo couleur, il est envisageable que l'un de ces deux utilisateurs puisse assurer un rôle de relais vers l'utilisateur 3. Toutefois si l'énergie disponible du périphérique relais est trop faible, la diffusion d'une vidéo couleur n'est pas envisageable. Il est alors possible de proposer l'installation d'un composant de conversion noir et blanc sur le périphérique servant de relais afin qu'il diffuse une version du service compatible avec l'énergie disponible. Ainsi, afin d'assurer la continuité du service, le périphérique 2 convertit la vidéo avant de l'envoyer au périphérique 3. Il faut préciser que pour de tels périphériques, les transmissions de données sont beaucoup plus coûteuses en énergie que le calcul [1]. La conversion de la vidéo en noir et blanc permet de réduire le nombre de données à transmettre et donc de préserver l'énergie des périphériques. De plus, délocaliser une partie des services permet d'équilibrer la charge du réseau et des périphériques. Bien évidemment, une telle solution requiert une connaissance de toute l'application en termes de services afin de choisir le meilleur déploiement. Notre approche utilise la reconfiguration et le redéploiement dynamique permettant de proposer des solutions satisfaisantes d'un point de vue infrastructure et QdS.

La Section 2 présente un bref état de l'art. La Section 3 présente la définition du contexte et de la qualité de service que nous utilisons dans cet article. Dans la Section 4, nous présentons notre modèle de QdS s'intéressant à l'utilité et à la pérennité d'une application. La Section 5 présente Kalimucho, notre plateforme de reconfiguration, et détaille l'heuristique de choix d'un déploiement. Enfin, la Section 6 détaille les aspects techniques et les tests de Kalimucho avec les capteurs SunSpots et un téléphone Android. Nous concluons par la Section 7 et présentons les travaux futurs.

II. ETAT DE L'ART

Le protocole de routage est la solution la plus répandue lorsqu'il s'agit de gérer la QdS des systèmes distribués ou mobiles [6][11]. Les protocoles de routage permettant de garantir une découverte et une mise en place de routes efficaces en termes de continuité et de bande passante. Cependant, ces protocoles ne suffisent pas à faire face à l'hétérogénéité et aux changements de contexte. Nous devons les associer à des mécanismes de routage de plus haut niveau.

A partir de ce constat, de nombreux travaux ont proposé des architectures logicielles pour gérer la QdS de ces systèmes contraints et mobiles. De telles approches abordent de façon globale la problématique de la prise en compte de l'application dans la mesure de la QdS. Music [14] propose un intergiciel sensible au contexte pour l'adaptation des systèmes mobiles. *"L'adaptation par planification réfère à la capacité de reconfiguration d'une application en réponse aux changements de contexte en exploitant les connaissances de sa composition et des méta-données de QdS associées à chacun des services la constituant"[8]*. Music considère que les applications sont développées avec un modèle de QdS tel que des fonctions d'utilité. Les applications sont décrites en différentes variants où les composants sont associés à des méta-données de QdS. Le processus de planification choisit les variantes de façon à maximiser l'utilité. Music est une approche centrée périphérique. Pour chaque adaptation, un domaine d'adaptation autour du périphérique définit une zone où l'adaptateur peut agir. De plus, chaque variante est associée à un plan de déploiement ce qui limite le nombre de solutions. QuAMobile [16] est une architecture logicielle générique pour la QdS dans les applications distribuées multimedia. Elle repose sur deux concepts : les composants spécifiques de gestion de QdS et service planner. Comme Music, le service planner permet de composer dynamiquement une configuration en function des critères de QdS. Cette approche montre l'intérêt de décrire les caractéristiques des composants d'une application et des périphériques pour fournir une configuration adéquate. Cependant aucun de ces travaux n'aborde le problème du déploiement. Des travaux comme Carisma et AxSeL [3] proposent des intergiciels prenant en compte le contexte au moment du déploiement pour des applications orientées services. Carisma propose un middleware pour l'adaptation d'applications mobiles reposant sur la capacité de réflexion des applications. La réflexion des applications permet une autonomie complète de la prise de décision et de la mise en œuvre de l'adaptation. Cependant, il ne permet de résoudre que des adaptations pour lesquelles le concepteur a prévu les situations et les comportements à mettre à œuvre. En revanche, il propose un mécanisme d'enchères intéressant pour éviter les conflits entre différentes politiques d'adaptation. AxSeL est une approche de chargement contextuel de service reposant sur OSGi. C'est une approche intéressante pour les applications orientées services où la cohérence d'une configuration repose en partie sur la correspondance entre les versions de services. AxSeL se sert des descripteurs de services et du contexte pour composer les déploiements adéquats aux besoins de l'application, aux conditions de ressources et aux préférences de l'utilisateur. Tout en essayant d'optimiser la consommation des ressources classiques, CPU, mémoire, énergie consommée par les composants, ces approches permettent de construire des configurations et leur déploiement à partir des dépendances physiques et logiques des services. Néanmoins, ces approches ne tiennent pas compte du coût en énergie des communications réseaux engendré par le déploiement des applications dans leurs décisions d'adaptation, coût non négligeable lorsqu'il est question de périphériques mobiles.

III. CONTEXTE ET QUALITE DE SERVICE

Il n'existe pas de définition unique du contexte. Le contexte est dépendant du domaine d'application. Dans cet article nous nous basons sur les définitions de Schilit et al. [15] et de David et al. [7]. Ces définitions montrent que le contexte et la QdS sont liés : les variations du contexte peuvent être vus comme une évolution de la QdS. Chacune des informations de contexte ayant un impact différent sur la QdS, nous définissons trois catégories de contexte : *utilisateur*, *utilisation* et *exécution*. Le *contexte utilisateur* réfère aux préférences de l'utilisateur, les services qu'il souhaite utiliser. Le *contexte d'utilisation* considère les contraintes d'utilisation. Ce sont les spécifications fonctionnelles de l'application, ce qu'elle peut ou ne peut pas faire. Le *contexte d'exécution* réfère au matériel

(CPU, mémoire, énergie) et aux capacités réseau. Ce sont les entités habituellement surveillées dans les systèmes sensibles au contexte. Le contexte est une mesure de la QdS d'une application. Traditionnellement utilisée dans les réseaux pour mesurer la performance des transmissions selon des critères quantitatifs comme le délai, la gigue ou le taux d'erreurs, la QdS ne peut plus être basée sur des critères matériels et réseaux [9] [17]. La prise en compte du point de vue des utilisateurs est nécessaire mais pas suffisante pour l'évaluation de la QdS lorsqu'il s'agit de périphériques contraints. En effet, l'utilisation de tels périphériques implique de devoir gérer la consommation énergétique et la façon de fournir les applications afin qu'elles correspondent aux contraintes du contexte et durent le plus longtemps possible.

Pour atteindre ce but, nous souhaitons agir à trois niveaux représentés dans la figure 2. Au niveau de l'infrastructure, nous souhaitons garantir à la fois la continuité de service et une durée de vie la plus longue possible de l'application malgré les variations des ressources matérielles et réseaux et la mobilité des périphériques. Au niveau de l'application, nous souhaitons garantir le respect des contraintes d'utilisation. Au niveau de l'utilisateur, nous souhaitons garantir le respect des souhaits de l'utilisateur et fournir une application utile.

Continuité de service. Le principal objectif de QdS est d'assurer un service continu en dépit des défaillances matérielles ou réseau et de l'hétérogénéité des appareils.

Pérennité. Nous souhaitons garantir la continuité de service des applications s'exécutant sur des appareils contraints. Une solution est de maximiser la durée de vie des applications. Un périphérique qui n'a plus d'énergie cause la déconnexion des services qu'il supporte compromettant la continuité de service. [1] montre que les transmissions réseaux coûtent plus d'énergie que les calculs. Ainsi, des solutions basées sur la mobilité des services minimisant les transmissions maximiseraient la durée de vie des applications.

Utilité. Nous définissons les contraintes d'utilisation comme les spécifications fonctionnelles d'une application que le système doit respecter. Par exemple, pour la visite du musée, le concepteur peut exprimer les contraintes suivantes :

- Si une conférence a lieu dans la pièce où se trouve le visiteur, éviter la fonctionnalité audio du service description pour ne pas perturber le visiteur.
- A la fermeture, active le service guidage vers la sortie du musée.

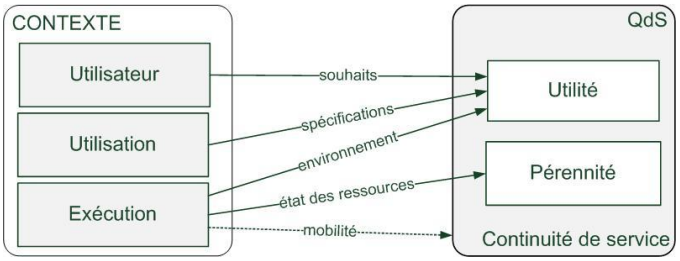

Figure 2. Interactions entre contexte et QdS

IV. UN MODELE DE QDS A DEUX DIMENSIONS

Afin de réaliser la décision de reconfiguration, nous devons mesurer la QdS de l'application. Nous souhaitons fournir une application utile et durable. Avec les systèmes mobiles, nous devons répondre aux besoins de l'utilisateur et maximiser la durée de vie de l'appareil. Pour minimiser la consommation d'énergie, la plupart des approches essaient de choisir les composants en fonction de leur consommation CPU et énergétique. Mais l'énergie est très liée à la distribution réseau.

L'originalité de notre approche est que notre gestion de l'énergie ne tient pas compte uniquement des ressources des appareils mais essaie également d'optimiser la charge réseau. Pour cela nous proposons un modèle de QdS permettant d'optimiser *l'utilité* et la *pérennité* d'une application.

A. *Utilité*

L'utilité est représentée par un classement de configurations. Chaque configuration a une note définie par le concepteur. Ce classement évolue en fonction du contexte. Les contraintes d'utilisation correspondent à des règles qui peuvent changer l'utilité d'un ensemble de configurations. Par exemple, quand le son est supérieur à 70dB ; nous évitons de proposer des configurations incluant le son pour éviter les désagréments à l'utilisateur. Ce type de contrainte peut être traduit comme une règle Evénement-Condition-Action :

If (sound > 70) throws *beginConference*

Event: [*beginConference*, sound, « - », 0.2]

Nous associons un événement à une fonctionnalité (son, vidéo, image etc.), un opérateur pour augmenter ou baisser la note d'utilité et un coefficient. Dans notre exemple, nous baissons la note des configurations incluant du son quand l'événement *beginConference* survient. Une telle règle modifie le classement pour placer les configurations concernées en dernière position.

B. *Pérennité*

Les fonctions d'utilité permettent de trouver une solution qui minimise l'impact de plusieurs critères dans un système [2]. Nous utilisons deux fonctions d'utilité pour évaluer la pérennité d'une application. La première permet de composer une configuration minimisant la consommation des ressources. La seconde permet de trouver un déploiement minimisant la charge réseau.

1) *Pérennité selon la consommation de ressources*

Chaque composant et périphérique est représenté par un triplet de notes : CPU consommé/disponible, mémoire et énergie (C, M, E). Les notes sont exprimées en pourcentage entre 0 et 1. Un composant qui ne consomme pas d'énergie a une note de 0 et 1 s'il consomme 100% de l'énergie du périphérique. Quand une configuration doit être déployée, nous calculons l'influence de chaque composant sur les périphériques. Pour un composant C déployé sur l'appareil H, nous obtenons la formule suivante :

$$(C\ on\ H) = (C_H - C_C, M_H - M_C, E_H - E_C)$$

Cependant, comment distinguer des composants qui sont équivalents ? Prenons trois composants, A(0.5, 0.2, 0.2), B(0.2, 0.5, 0.2) et C(0.2, 0.1, 0.6). Si nous faisons la moyenne des triplets, nous obtenons 0.3 pour tous les composants. Si nous déployons A sur le périphérique H, A consommera trop d'énergie et H sera rapidement hors service. B consomme beaucoup de mémoire et H ne pourra pas supporter d'autres composants. C consomme beaucoup de CPU qui peut empêcher les composants de fonctionner correctement. Pour résumer, le facteur discriminant d'un composant est celui qui a le plus d'impact lors du déploiement sur un périphérique. La note de pérennité selon les ressources (QoS_RC) est alors le min des trois notes :

$$QoS\_RC(C\ on\ H) = max(0, min(C_H - C_C, M_H - M_C, E_H - E_C))$$

Si nous déployons plusieurs composants sur un périphérique, nous faisons la somme des notes :

$$QoS\_RC(A,B\ on\ P) = max(0, min(C_P - C_A - C_B, M_P - M_A - M_B, E_P - E_A - E_B))$$

Finalement, nous calculons la pérennité d'une configuration par la formule suivante :

$$QoS\_RC(configuration) = \min(QoS\_RC\ (A,B\ on\ P), QoS\_RC\ (C\ on\ H))$$

Ce calcul n'est pas le plus optimal puisqu'on préfère choisir une configuration qui consomme peu d'énergie sur deux périphériques au lieu d'une qui consomme beaucoup d'énergie sur un périphérique. Nous préférons épuiser peu à peu les appareils plutôt que un à un. Le but étant de maximiser la durée de vie des appareils, nous évitons de mettre des appareils hors service.

2) Pérennité selon le coût réseau

Nous calculons la pérennité selon la consommation réseau (QoS_NC) de la même manière pour pour la pérennité selon les ressources. Pour un déploiement choisi, nous connaissons les périphériques supportant chacun des composants et nous connaissons les liens réseaux entre chaque périphérique (à partir des dépendances entre composants).

Dans de précédents travaux, nous avons proposé une méthode de conception où chaque composant et périphérique sont décrit dans une carte d'identité [12]. Dans cette carte d'identité, un périphérique connait le débit théorique (BWTh) de chacun des réseaux qu'il peut atteindre (WiFi, Bluetooth, Zigbee, etc.). Dans nos applications, les composants et les flux de données sont encapsulés dans des containers (Osagaia pour les composants, Korrontea pour les flux de données). Ces containers sont composés de trois entités, Unité d'Entrée, Unité de Sortie et Unité de Contrôle, qui sont la principale source d'information de la plateforme. Quand une unite détecte une variation de son contexte, elle informe la plateforme qui récupère les informations nécessaires (section 5). Grâce aux containers Korrontea, nous pouvons connaitre le débit des connecteurs pendant l'exécution et calculer le débit moyen de chaque connexion. Le débit moyen entre les appareils H_1 et H_2 est donné par:

$$BW_{H_1H_2} = BWTh_{H_1H_2} - BWAv_{H_1H_2}$$

Enfin, les cartes d'identité des composants indiquent le débit de sortie produit. Une méthode simple de connaître le débit de sortie d'un périphérique est de faire la somme des débits produits par les composants qu'il supporte.:

$$QoS\_NC = max\left(0, \frac{BW_{H_iJ} - \sum^{output\ bandwidth\ of\ components\ between\ H_iH_j}}{BWTh_{H_iH_j}}\right)$$

3) Evaluation de la QdS

Nous pouvons représenter les applications par des graphes de configuration [10] où une application peut être réalisée par une ou plusieurs configurations de QdS différentes Pour chaque application, les configurations sont classées par ordre décroissant d'utilité, base de l'évaluation.

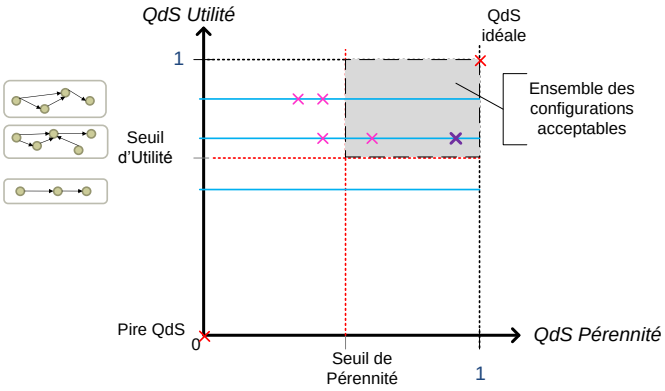

Figure 3. QoS evaluation.

Nous représentons la QdS d'une configuration (et son déploiement) dans un diagramme à deux dimensions où l'axe des abscisses représente la Pérennité et l'axe des ordonnées représente l'Utilité. Nous plaçons des seuils de QdS Utilité et Pérennité permettant de définir un ensemble des configurations de QdS satisfaisante (fig. 3). Ces seuils sont définis tout d'abord à la conception de l'application et peuvent être modifiés en fonction du contexte afin d'agrandir l'ensemble des configurations acceptables et trouver un déploiement adéquat. Nous étudions chaque configuration depuis le haut du classement d'utilité et nous calculons la pérennité de chacun de leur déploiement. Tant que la QdS est hors des limite, nous étudions le déploiement suivant ou la configuration suivante dans le classement jusqu'à trouver un déploiement qui corresponde aux critères de QdS. S'il existe plusieurs déploiements valables, celui de meilleur QdS est sélectionné (fig. 3).

TABLE I. CLASSEMENT DES PERIPHERIQUES

Device	**H1**	**H2**	**H3**	**A**	**C**	**B**	**D**	**F**	**E**
Impact	0	0	0	1	1	1	1	1	2
Type	Fixed	CDC[2]	CDC	Fixed	Fixed	CDC	CLDC[2]	CLDC	CLDC
Energy	-	1	0.8	-	-		0.2	0.3	
CPU				0.9	0.75				

Le paragraphe suivant présente Kalimucho, une plateforme de reconfiguration qui implémente le modèle de QdS à travers une heuristique de déploiement contextuel.

V. KALIMUCHO

Nous avons choisi de nous intéresser à la gestion de la QdS dans les systèmes mobiles et contraints par reconfiguration dynamique des applications. Pour cela, nous utilisons des applications basées composants permettant de fourni la flexibilité et la modularité nécessaires aux reconfigurations. Les applications sont construites à partir du modèle de composant Osagaia et du modèle de connecteur Korrontea [4]. Ces modèles encapsulent les composants/connecteurs dans des containers permettant de capturer le contexte (activité, QdS, délai, etc.) et de superviser le cycle de vie des composants/connecteurs (start, stop, connexion, déconnexion, migration). Ces containers peuvent être contrôlés par la plateforme via l'Unité de Contrôle. Ainsi nous pouvons agir directement sur la composition et la distribution des applications lors des reconfigurations. Ces containers sont constitués de deux unités de communication : l'Unité d'Entrée et l'Unité de Sortie. Ces unités, dotées de buffers, permettent de superviser les communications entre composants et de détecter les anomalies telles que la famine ou la saturation de données dans les buffers. C'est une source d'information utile pour la gestion de la mobilité de composants. De plus, l'utilisation de buffers nous a permis de pouvoir garantir une continuité de service puisqu'il évite la perte de données lors des migrations de composants.

Pour gérer la QdS de ces applications, nous proposons la plateforme de reconfiguration Kalimucho[1]. Kalimucho consiste en cinq services collaboratifs, distribués sur tous les périphériques afin d'obtenir une connaissance globale du système. Cette plateforme est capable de répondre aux variations du contexte en modifiant la structure et le déploiement de l'application en utilisant cinq actions de base : ajouter, supprimer, connecter, déconnecter et migrer un composant/connecteur. Les cinq services de Kalimucho permettent d'assurer les objectifs suivants : (1) Capturer le contexte. C'est le service Supervision. (2) Proposer un déploiement maximisant l'Utilité et la Pérennité de l'application. Ce service assure le calcul d'évaluation de la QdS des configurations. C'est le service Reconfiguration. (3) Rechercher des routes valides afin de maintenir la continuité de service. C'est le service Routage. (4) Gérer l'hétérogénéité des appareils. Les composants et les connecteurs doivent pouvoir être exécuté et mis en place sur n'importe quel périphérique. Pour cela la Fabrique de Conteneurs et la Fabrique de Connecteurs permettent de créer des containers de composants et de connecteurs adaptés aux caractéristiques de chaque périphérique. L'ensemble de ces services est détaillé dans [13].

En fonction des éléments de contexte capturés par le service Supervision, Kalimucho propose deux processus de reconfiguration. Soit elle peut migrer un service : elle essaie de redéployer de façon différente les composants du service courant. Soit elle déploie une nouvelle configuration du service : elle évalue un ensemble de configurations respectant l'Utilité requise et essaie de trouver un déploiement respectant le critère de Pérennité.

Pour trouver un tel déploiement, Kalimucho implémente une heuristique de déploiement contextuel.

A. Heuristique pour un déploiement contextuel

Le but de cette heuristique est de trouver une configuration et le déploiement associé qui satisfassent les conditions de contexte et les critères de QdS : utilité et pérennité.

Pour limiter la consommation énergétique, nous proposons de limiter le nombre de liaisons réseaux. Sachant qu'il existe des dépendances de localisation pour certains composants, nous essayons de grouper les autres composants sur les périphériques imposés. Quand ils ne peuvent plus supporter de composants supplémentaires, nous essayons de placer les composants restants sur les périphériques se situant sur les chemins entre les périphériques imposés (fig. 6).

Cette approche repose sur un classement des périphériques (table 1) et des composants qui repose lui-même sur deux concepts : le poids d'un composant et le poids d'un périphérique. Les configurations peuvent être représentées par un graphe orienté (fig. 4). Nous définissons ensuite des chemins entre les composants fixés à des périphériques particuliers (dépendances de localisation). Nous définissons le poids d'un composant par son rang minimal dans les chemins précédemment définis auxquels il appartient. Le poids représente la distance minimale entre ce composant et un composant fixé sur un périphérique. Comme les composants fixés sur des périphériques sont toujours au début d'un chemin (par construction), ils ont toujours un poids de 0. Nous appliquons la même définition aux périphériques sauf que le réseau est représenté par un graphe non-orienté et donc les chemins sont bidirectionnels (fig. 5).

A partir de ce classement, nous essayons de déployer chaque configuration jusqu'à trouver un déploiement qui réponde aux critères de QdS. Afin d'accélérer cette recherche, nous améliorons ce classement par quatre autres critères : type

[1] *https://kalimucho.dev.java.net*

d'un périphérique, énergie disponible, CPU disponible et mémoire disponible (table 1).

- ***Type de périphérique***: Ce classement permet d'utiliser le maximum de périphériques non limités. Nous distinguons trois types de périphériques : Fixe, CDC[2] et CLDC. Les périphériques CLDC sont le plus limités (téléphone, capteur).
- ***Energie***: L'énergie disponible sur les périphériques est un critère important puisqu'il adresse directement la pérennité de l'appareil et donc celle de l'application. Il implique a priori plus de reconfiguration que le CPU ou la mémoire.
- ***CPU***: La charge CPU est un critère moins important que l'énergie. Cependant, une charge CPU trop élevée peu ralentir le fonctionnement des composants.
- ***Mémoire***: La mémoire disponible n'a pas d'impact sur la pérennité au moment de déploiement. Elle influence les futurs ajouts de composants sur le périphérique.

A partir de ce classement, l'heuristique utilise une approche récursive pour calculer le placement de chaque composant :

1. Nous classons les composants dans une liste CL selon leur poids.
2. Nous sélectionnons le premier composant de poids ==1
    a. S'il n'en existe aucun, cela signifie que tous les composants ont un poids == 0 et qu'ils sont tous placés. L'évaluation se termine avec succès, il ne reste plus qu'à déployer la configuration
3. Nous classons les périphériques dans une liste DL, selon les critères de la table 1.
4. Nous évaluons l'incidence du placement du composant choisi sur le premier périphérique de DL.
    a. Si (*QoS_RC >= seuil*) alors le placement de ce composant sur ce périphérique est retenu. Si le placement retenu concerne le dernier composant de la liste CL, alors nous pouvons évaluer l'incidence du déploiement trouvé sur le réseau (QoS_NC). Si (*QoS_NC >= seuil*), le placement de ce composant sur ce périphérique est retenu. On considère alors ce composant comme placé, et on appelle récursivement l'heuristique à l'étape 1
    b. Sinon, nous revenons en (4) avec le périphérique suivant de LP s'il en reste, sinon l'appel récursif en cours se termine par un échec qui provoquera une nouvelle tentative pour le composant précédemment placé.

En cas d'échec, nous pouvons mettre à jour les seuils de QdS pour pouvoir sélectionner de nouvelles configurations. Si cette mise à jour ne permet pas de trouver un déploiement satisfaisant et que toutes les configurations ont été testées, cela signifie que l'application ne peut pas être adaptée. Un simulateur a été développé afin de tester le fonctionnement de cette heuristique. Un exemple de déploiement testé est celui de la configuration "Text" représentée sur la figure 4 sur le réseau de la figure 5. La configuration "Text" est composée de 8 composants, C_1 à C_8. Des dépendances de localisation ont été définies: C_1 est placé sur H_1, C_5 et C_6 sont placés sur H_2 et C_8 est placé sur H_3. Le résultat de l'heuristique sur la figure 6 confirme que les composants sont groupés sur les périphériques déjà utilisés (H1, H2 et H3) afin de limiter la consommation d'énergie.

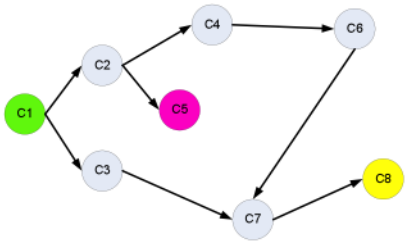

Figure 4. Graphe de la configuration "Text"

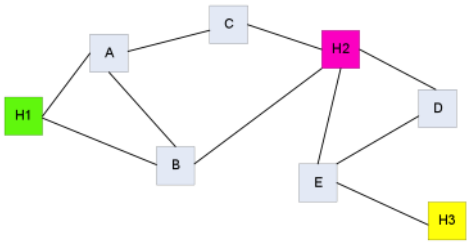

Figure 5. Graphe du réseau

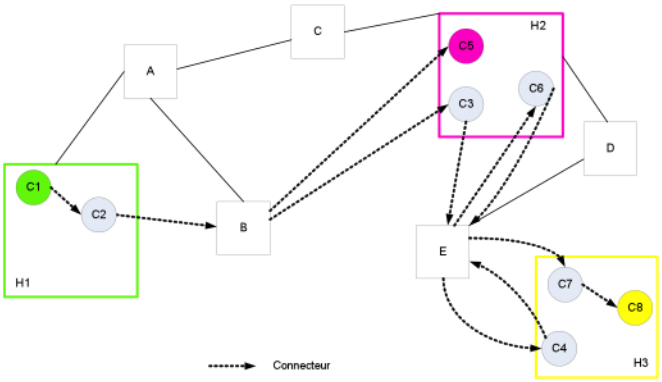

Figure 6. Résultat du déploiement de la configuration "Text"

D'après les différents tests réalisés sur le simulateur, visant des applications de petites tailles, l'heuristique parvient à trouver un déploiement en quelques millisecondes. Dans le cas de configuration réseau contraintes (existences d'îlots), le calcul nécessite quelques secondes. Enfin, bien que l'heuristique ne comporte pas de mécanismes de vérification du coût de redéploiement, les tests réalisés sur différentes configurations ont révélé que le classement utilisé permet d'obtenir à chaque redéploiement des distributions proches n'impliquant que peu de migration de composants, limitant ainsi les transferts réseau, et donc le coût énergétique.

[1] Selon les descriptions de types de périphérique de Sun Microsystem

VI. EXPERIMENTATIONS

Nous avons développé un prototype de Kalimucho s'exécutant sur un ordinateur portable, un Smartphone (Android) et deux capteurs SunSpot. Cette implémentation de Kalimucho permet de déployer et de reconfigurer des applications basées composants à partir de fichiers de commandes. Elle permet également de capturer les éléments de contexte nécessaires à la prise de décision. Le service Supervision rapporte, sous forme d'une alarme, les changements de contexte et choisit l'adaptation que la plateforme doit appliquer. Pour cela la service Supervision est capable de lire l'état d'un périphérique (mémoire, CPU, batterie), Lire l'état d'un composant ou d'un connecteur (QdS, activité, connexions, etc.) et Relayer des états vers d'autres plateformes. Le service Reconfiguration est capable de créer ou supprimer des composants/connecteurs, de migrer des composants, de connecter ou déconnecter une entrée de composant et de supprimer ou dupliquer une sortie de composant et d'envoyer des commandes aux autres plateformes. Une version de Kalimucho a été développée pour chaque type de périphérique : Kalimucho pour les périphériques fixes tels que les ordinateurs portables (260Ko); Kalimucho pour les périphériques CDC tels que les Smartphones (356Ko); Kalimucho pour les périphériques CLDC tels que les capteurs Sun Spot (169Ko). Enfin, les périphériques n'utilisant pas tous le même réseau, nous fournissons un outil permettant de relayer l'information à travers différents réseaux. Dans notre prototype, le PC et le Smartphone utilisent le WiFi alors que les capteurs utilisent le ZigBee. Pour que les capteurs puissent communiquer avec le reste des périphériques, une station-base est connecté au PC pour agir comme une passerelle.

Nous proposons un scenario utilisant les cinq actions de la plateforme : déployer, remplacer, ajouter, migrer un composant et reconfigurer un service. Nous mesurons le temps d'exécution de chaque action.

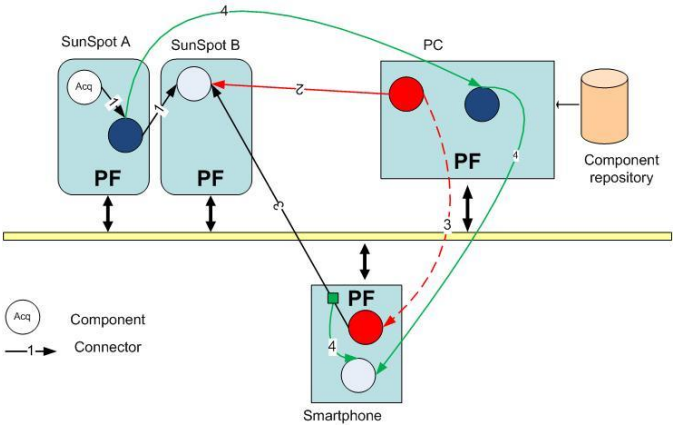

Figure 7. Display application

Une application d'affichage de l'inclinaison des capteurs composée de 3 composants, capture, traitement et affichage, est déployée sur les capteurs (fig. 7). (1) Nous remplaçons le composants de traitement afin d'inverser l'affichage. (2) Nous ajoutons ensuite un composants sur le PC pour choisir la couleur d'affichage. (3) Nous devons nous assurer que le Smartphone peut communiquer avec le capteur via le PC. (passerelle). (4) Enfin, nous reconfigurons l'application afin de visualiser l'affichage à la fois sur le Smartphone et le capteur. La Table 2 résume les temps d'exécution de chacune des commandes utilisées dans le scenario. Nous pouvons remarquer que ces temps d'exécution sont tous de l'ordre de la milliseconde, ce qui est acceptable pour de tels périphériques contraints.

TABLE II. EXECUTION TIME OF KALIMUCHO COMMANDS ON A SUNSPOT SENSOR AND ON A NEXUS ONE MOBILE PHONE

Command	Execution time in ms	
	SunSpot	***Nexus One***
Création d'un composant	70 to 170 ms	450 to 750 ms (téléchargement d'un code de 2Ko)
Suppression d'un composant	20 ms minimum. lié au temps nécessaire au composant pour se terminer	15 ms. Dépend de l'activité du composant
Création d'un connecteur	Interne: 70 to 110 ms Réparti: 100 to 190 ms sur l'appareil recevant la commande, 30 to 120 ms sur l'autre	Interne: 3 to 15 ms Réparti: 10 to 100 ms
Suppression d'un connecteur	Interne: 60 to 80 ms Réparti: 100 to 260 ms sur l'appareil recevant la commande, 30 to 120 ms sur l'autre	Interne: 3 to 15 ms Réparti: 3 to 25 ms
Déconnexion ou reconnexion d'une entrée	20 to 60 ms	2 to 7 ms
Duplication d'une sortie	20 to 80 ms	2 to 7 ms
Lecture de la QdS d'un conteneur	80 ms	
Lecture de l'état d'un conteneur	70 to 80 ms	
Lecture de l'état d'un périphérique	70 to 90 ms	
Migration d'un composant	90 to 230 ms	650 à 750 ms (téléchargement d'un code de 2Ko)

VII. CONCLUSION ET TRAVAUX FUTURS

L'informatique pervasive est devenue une réalité. A présent, les utilisateurs souhaitent utiliser leurs applications préférées depuis n'importe quel périphérique. Dû à la mobilité des appareils, les utilisateurs souhaitent que les applications s'adaptent automatiquement en fonction du contexte environnant. Ceci nous impose de faire face à de nouveaux défis. Les applications doivent à présent être sensibles au contexte à cause de la limitation des ressources des appareils et de la variabilité du contexte d'exécution [18]. La plupart des approches proposées traitent le problème de la consommation énergétique par l'adaptation dynamique des applications et de leur déploiement. Néanmoins, elles considèrent uniquement la consommation en énergie, CPU et mémoire. La consommation énergétique étant liée au nombre de transferts réseaux dans le cas des périphériques mobiles, nous proposons une approche incluant la considération du coût des communications.

Nous proposons alors un modèle de qualité de service permettant de garantir l'utilité d'une application et de maximiser sa durée de vie. La pérennité d'une application est une notion fondamentale avec l'utilisation d'appareils mobiles puisque une application grande qualité est inutile si elle ne dure que quelques secondes. L'utilité mesure l'adéquation de l'application fournie avec les besoins de l'utilisateur et les spécifications de l'application tandis que la pérennité mesure la durée de vie d'une application en fonction de la consommation des ressources et du réseau. Nous proposons ensuite Kalimucho, une plateforme de reconfiguration dynamique des applications mobiles. Elle implémente le modèle de QdS à travers une heuristique récursive de déploiement contextuel des applications. Enfin, nous avons testé Kalimucho sur différents périphériques tels qu'un ordinateur portable, un Smartphone, et des capteurs. Les résultats montrent que le temps d'exécution des actions de reconfiguration est acceptable pour des périphériques si limités.

Cependant, ces tests ne considèrent que des applications et des réseaux de petites tailles. Les travaux futurs visent à étudier le passage à l'échelle d'une telle approche, notamment en termes de temps de recherche d'une solution. De plus, les calculs de QdS sont basés sur des mesures statiques de consommations des ressources par les composants. Enfin, l'heuristique de choix d'un déploiement n'évalue que la QdS du service pour lequel un évènement de reconfiguration a été déclenché et non pour toute l'application. Quand nous reconfigurons, nous offrons la possibilité d'obtenir la meilleure QdS pour un service. La reconfiguration d'un service n'implique pas la reconfiguration des autres services supportés par le périphérique. Une reconfiguration entrainant des modifications du contexte d'exécution, elle peut entrainer de nouvelles reconfigurations pour d'autres services.

Les travaux futurs se focalisent sur la conception et les tests de l'heuristique de déploiement contextuel. Par ailleurs, tous les périphériques ne sont pas dotés des mêmes dispositifs. Ainsi, par exemple, tous ne disposent pas d'un GPS. La possibilité pour la plateforme de déployer des services, peut permettre à un utilisateur dépourvu de GPS, de se localiser géographiquement du seul fait qu'il se situe à proximité d'un dispositif qui en est doté et sur lequel la plateforme a installé un service à cet effet. De plus, la composition de services peut permettre la réalisation de services, non initialement prévus par le concepteur de l'application, mais nécessaires sur le moment à l'utilisateur. Actuellement, la plateforme Kalimucho ne permet pas une telle modularité. Les services sont figés à la conception dans des configurations pour plus de facilité lors de la recherche de la configuration à déployer en cas de reconfiguration. Proposer une telle plateforme permettrait de proposer de nouvelles fonctionnalités aux utilisateurs et, par conséquent, d'accroître les possibilités d'offrir une qualité de service suffisante à ces derniers sans figer les configurations. L'idée est de proposer un modèle de plateforme, modulable, à la façon d'un puzzle, qui permettrait d'une part, de déployer une plateforme adaptée aux capacités du périphérique et, d'autre part, de construire à la volée la plateforme adaptée au contexte d'utilisation.

Les auteurs remercient l'ANR pour le soutien financier de ces travaux dans le cadre du projet MOANO.

BIBLIOGRAPHIE

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. "Wireless sensor networks : a survey. Computer Networks", 38(4) :393–422, 2002.

[2] Mourad Alia, Viktor S. Wold Eide, Nearchos Paspallis, Frank Eliassen, Svein O.Hallsteinsen, and George A. Papadopoulos. "A utility-based adaptivity model for mobile applications". In AINA Workshops, pages 556–563. IEEE Computer Society,2007

[3] Ben Hamida, Le Mouel, Frénot, and Ben Ahmed. "A graph-based approach for contextual service loading in pervasive environments". In *Proceedings of the 10th International Symposium on Distributed Objects and Applications (DOA'2008)*, Lecture Notes in Computer Science, vol. 5331, pp. 589–606, Springer Verlag, 2008.

[4] Bouix, Roose, and Dalmau." The korrontea data modeling". In Ambi-Sys'08 : Proceedings of the 1st international conference on Ambient media and systems, pages 1–10, ICST, 2008.

[5] Chakeres, I. D., Belding-Royer, E. M.: "AODV Routing Protocol Implementation Design*"*. In Proceedings of the International Workshop on Wireless Ad Hoc Networking, WWAN'04 Springer, 2004.

[6] Chen, T. w., Tsai, J. T., and Gerla Y. "Qos routing performance in multihop, multimedia, wireless networks". In In Proceedings of IEEE International Conference on Universal Personal Communications, ICUPC, pages 557–561, 1997.

[7] David, P., C., and Ledoux, T.: "Wildcat: a generic framework for context-aware applications". In Sotirios Terzis and Didier Donsez, editors, MPAC, volume 115 of ACM International Conference Proceeding Series, pages 1–7. ACM, 2005

[8] Floch, Hallsteinsen, Stav, Eliassen, Lund, and Gjørven. "Using architecture models for runtime adaptability". IEEE Software, 23(2) :62–70, 2006.

[9] Franken, L. J. N., Haverkort, B. R.: "Quality of service management using generic modelling and monitoring techniques". Distributed Systems Engineering. 4, 1 (1997), 28–37

[10] Laplace, S., Dalmau, M., Roose, P.: Kalinahia: "Considering quality of service to design and execute distributed multimedia applications", In NOMS, pages 951-954IEEE, 2008.

[11] Lin, C. R., Liu, J. S., "Qos routing in ad hoc wireless networks". IEEE Journal On Selected Areas In Communications, 17(8) :1426–1438, 1999.

[12] Louberry, C., Roose, P., Dalmau, M.: "QoS Based Design Process for Pervasive Computing Applications", ACM Mobility 2009, Nice, France (2009)

[13] Louberry, C., Dalmau, M., Roose, P. "Software Architecture for Dynamic Adaptation of Heterogeneous Applications", In NOTERE '08, Proceedings of the 8th international conference on New technologies in distributed systems, 2008.

[14] Rouvoy, Barone, Ding, Eliassen, Hallsteinsen, Lorenzo, Mamelli, and Scholz. "Music : Middleware support for self-adaptation in ubiquitous and service-oriented environments". In Software Engineering for Self-Adaptive Systems, volume 5525 of Lecture Notes in Computer Science, pages 164–182. Springer, 2009.

[15] Schilit, B., N., and Theimer, M, M.: "Disseminating active map information to mobile hosts". IEEE Network, 8(5): 22–32, 1994

[16] Sten Lundesgaard Amundsen, Ketil Lund, Carsten Griwodz, and P° al Halvorsen. "Qos-aware mobile middleware for video streaming". In EUROMICRO-SEAA, pages 54–61.IEEE Computer Society, 2005.

[17] Vogel, A., Kerhervé, B., von Bochmann, G., Gecsei J.: "Distributed multimedia and qos : A survey. IEEE MultiMedia". 2, 2, 10–19, 1995

[18] Zheng D., Wang J., Jia Y., Han W., Zou P. 2007. "Deployment of Context-Aware Component-Based Applications Based on Middleware". UIC. Volume 4611 of Lecture Notes in Computer Science. Springer.