Intégration de périphériques contraints par reconfiguration dynamique d'applications - Cas des capteurs sans-fil

Christine Louberry, Marc Dalmau, Philippe Roose

Intégration de périphériques contraints par reconfiguration dynamique d'applications

Cas des capteurs sans-fil

Christine Louberry — Marc Dalmau — Philippe Roose

LIUPPA – IUT de Bayonne
2 allée Parc Montaury
64600 Anglet
{prenom.nom}@iutbayonne.univ-pau.fr

RÉSUMÉ. *L'apparition récente de capteurs mobiles sans-fil capables de rendre compte de leur environnement physique et d'effectuer des traitements, permet d'envisager des applications capables de prendre en compte leur contexte environnemental et de réagir en fonction de ses évolutions. Ceci suppose de pouvoir concevoir des applications intégrant à la fois des composants logiciels et matériels, capables de communiquer et d'utiliser les informations contextuelles disponibles. Ces travaux s'intéressent à l'intégration de périphériques mobiles et en particulier de capteurs sans-fil dans les applications distribuées. Nous proposons que les problèmes soulevés par les contraintes liées à ces composants (mobilité, faiblesse des ressources, etc.) soient résolus par reconfiguration dynamique des applications. Dans ce but nous présentons un modèle de composants unifié indépendant de leur nature matérielle ou logicielle ainsi qu'une plate-forme de supervision à services permettant de gérer et de reconfigurer des telles applications.*

ABSTRACT. *The recent emergence of mobile wireless sensors able to give information on their physical environment and to process data, should enable to provide applications aware of their physical context and to evolve according to its changes. This requires to be able to design applications that integrate both software and hardware components, able to communicate and to use available contextual information. These works are concerned with the integration of mobile peripherals and particularly wireless sensors in distributed applications. The problems introduced by these components (mobility, resources etc.) can be resolved by dynamic reconfiguration of the applications. So we present a unified component model independent of their hardware or software nature and a service based platform to manage and reconfigure such applications*

MOTS-CLÉS : *Applications distribuées, modèle de composant, architecture logicielle, reconfiguration, capteurs sans fils.*

KEYWORDS: *Distributed applications, component model, software architecture, reconfiguration, wireless sensors.*

1. Introduction

Face à la demande grandissante pour des services de plus en plus riches et personnalisés, le défit aujourd'hui est de proposer des applications qui s'adaptent tant aux souhaits de l'utilisateur qu'à l'environnement réel.

On rencontre, à l'heure actuelle, deux grandes familles d'applications : celles qui sont conçues et développées pour fonctionner sur des machines classiques et celles pour des périphériques contraints.

Si dans le premier cas on considère généralement que les ressources disponibles ne sont pas critiques, il faut, tout au contraire, dans le second tenir compte des performances disponibles, des capacités de mémoire etc.

L'arrivée massive de périphériques nomades permet d'envisager des applications utilisant ces dispositifs en même temps que des machines plus classiques. En effet les utilisateurs souhaitent disposer sur leurs PDA et téléphones portables de services équivalents à ceux qu'ils ont l'habitude d'utiliser. En outre, en raison de la mobilité naturellement induite par ces dispositifs, la qualité du service offert ne peut pas faire abstraction du contexte environnemental. La récente mise sur le marché de capteurs sans fil (*smart sensors*) dotés de processeurs et de moyens de communication permet d'imaginer l'intégration de ces dispositifs aux applications proposées afin de disposer d'informations sur l'environnement d'exécution de ces applications.

L'introduction de périphériques légers dans de telles applications se heurte aux problèmes liés à leur mobilité, à leur faibles ressources, aux débits limités de communication qu'ils offrent et enfin, pour la majorité d'entre eux, aux contraintes d'énergie des batteries.

La grande majorité des travaux liés aux réseaux de capteurs concernent l'optimisation des ressources matérielles (capacité de calcul, énergie) et réseau (contrôle de congestion, agrégation des données, etc.). Toutefois il s'agit généralement de solutions ad-hoc ne pouvant pas s'étendre à l'ensemble des composants de l'application. Les capteurs y sont utilisés pour leurs fonctions propres de mesure de l'environnement et leur capacité à transmettre et relayer l'information, en veillant à en maximiser la durée de vie. Il existe actuellement peu de recherches sur l'intégration de tels dispositifs dans des environnements hétérogènes où collaborent composants logiciels et capteurs.

L'approche que nous présentons dans cet article est centrée sur le domaine des architectures logicielles permettant l'intégration de capteurs sans fil et, de façon générale, de périphériques contraints. Nous proposons une solution se basant sur la reconfiguration dynamique de ces architectures par redéploiement des composants métier. Ainsi, la mobilité et les problèmes liés aux faibles débits de communication peuvent être résolus en rapprochant les composants de traitement de la source des données qu'ils traitent. Tandis que les contraintes d'énergie et de ressources seront

résolues en déplaçant des composants métier de façon à soulager certains périphériques.

Cette reconfiguration est supervisée par une plate-forme répartie qui, à partir des informations qu'elle obtient des composants de l'application (charge, débits, etc.), choisit de nouvelles configurations à déployer.

Il est, de plus, intéressant de ne plus considérer les capteurs simplement comme des dispositifs capables d'effectuer des mesures. En effet, lorsqu'ils n'utilisent pas la totalité de leur mémoire et de leur capacité de calcul, il leur est possible d'héberger d'autres composants logiciels en relation ou non avec leurs fonctions propres. Ainsi les capteurs participent à l'infrastructure matérielle des applications en offrant de nouvelles possibilités d'hébergement de fonctionnalités. Ceci permet de proposer de nouvelles configurations pour les applications et, par conséquent, d'accroître les possibilités d'offrir une qualité de service suffisante.

La section 2 présente le modèle de composants Osagaia sur lequel se base notre modèle unifié. La section 3 présente un modèle adapté aux capteurs sans-fil actuellement sur le marché et détaille leur intégration dans le modèle Osagaia. La section 4 présente les bases de l'architecture logicielle initialement proposée pour la reconfiguration des applications ainsi qu'une implémentation de la plate-forme de reconfiguration appelée Kalinahia. La section 5 présente l'évolution de la plate-forme Kalinahia adaptée à une implémentation d'applications hétérogènes. La section 6 présente les implémentations développées sous forme de prototypes. La section 7 propose une discussion sur les travaux apparentés à ces recherches dans le domaine des architectures logicielles pour les réseaux de capteurs. Enfin, la section 8 conclut cet article et présente les travaux à venir.

2. Le modèle Osagaia

Le modèle Osagaia [4] a été proposé pour réaliser la partie applicative. L'objectif était de fournir des solutions permettant de réaliser simplement une application multimédia distribuée par interconnexion de composants métier pouvant être supervisés par une plate-forme. Ce modèle s'intéresse particulièrement aux problèmes de synchronisation des flux et de connexion et déconnexion dynamique des composants. Il est constitué de deux entités :

- Le Conduit, jouant le rôle de connecteur, assure le transport synchrone des flux multimédias de l'application. Il peut être distribué sur l'Internet.
- Le Processeur Élémentaire (PE) est un conteneur qui fournit un environnement d'exécution pour le Composant Métier (CM). Le CM implémente un traitement multimédia particulier. Par exemple, un CM de capture vidéo implémente le mécanisme nécessaire pour fournir la capture.

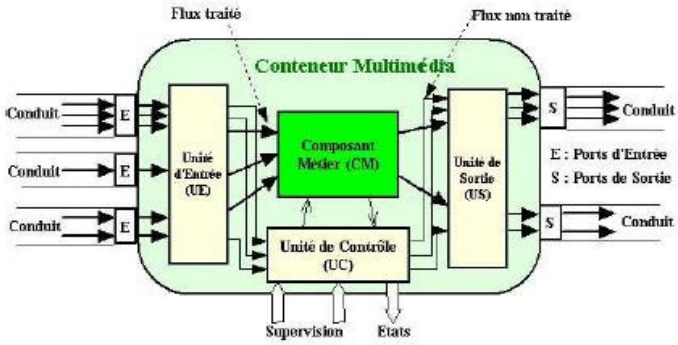

Figure 3 : *Architecture interne du Processeur Élémentaire*

Les entités du modèle sont connectées par des ports d'entrée/sortie. Les ports sont le moyen par lequel circulent les flux multimédias d'un PE vers un Conduit (connecteur) puis de ce Conduit vers un autre PE. Les ports acceptent des échantillons en entrée et fournissent des échantillons en sortie. Le port est une unité structurelle de connexion entre les différentes entités du modèle (élément connectable).

Le Composant Métier implémente un traitement multimédia particulier (implémentation fonctionnelle). Le CM, exécuté dans le PE, est dirigé par les données, c'est-à-dire qu'il ne peut s'exécuter que s'il dispose de données dans le port d'entrée du PE qui l'encapsule.

Le PE assure les propriétés non-fonctionnelles pour une exécution correcte du CM (propriétés fonctionnelles) et de toute l'application. Il est composé d'une unité d'entrée (UE), d'une unité de sortie (US) et d'une unité de contrôle (UC) comme le montre la figure 3. Le PE est supervisé par la plate-forme (ajout, suppression ou déplacement). Il possède des ports d'entrée/sortie pour chaque flux multimédia entrant ou sortant. Ces ports lui permettent de se connecter aux Conduits. Chaque port est lié respectivement à l'UE ou l'US. Ces unités constituent les interfaces entre le CM et les flux multimédias. Elles offrent au CM des outils pour lire (respectivement écrire) dans les ports d'entrée (respectivement sortie). L'UC gère tous les éléments du PE. Le cycle de vie du CM est piloté par l'UC au moyen des méthodes *init()*, *start()* et *stop()*. Elle contrôle également la circulation de l'information au travers du PE. En particulier, l'UC assure que les flux entrants qui ne nécessitent pas de traitement restent synchrones entre eux et avec les flux traités.

Le conduit possède également une unité d'entrée, une unité de sortie et une unité de contrôle. Son rôle est de transmettre des flux d'information de l'entrée vers la sortie en utilisant éventuellement un réseau tout en leur appliquant des politiques de gestion (conservation de la synchronisation, temps réel, priorités …)

3. Evolution du modèle Osagaia

Afin de concevoir des applications réactives à leur contexte environnemental, nous avons choisi d'intégrer des capteurs sans-fil capables de mesurer et d'exploiter diverses quantités physiques. Jusqu'à présent, il existe une séparation entre les fonctionnalités des capteurs et les services que rendent les applications les utilisant. En effet, à l'heure actuelle, la plupart des applications utilisant des capteurs sans-fil, s'intéressent uniquement à l'exploitation des données qu'ils mesurent. Il y a d'un part le réseau de capteurs et d'autre part l'application exploitant les données contextuelles recueillies [15]. Un serveur permet de centraliser les mesures et servir de passerelle entre les deux parties. Les capteurs sont utilisés uniquement pour leur fonction de mesure de l'environnement.

Nous souhaitons une plus forte implication des capacités des capteurs dans les applications. Nous souhaitons qu'ils fassent partie intégrante des applications.

Le modèle Osagaia présenté ci-dessus offre la possibilité d'intégrer des composants logiciels dans les applications et de les superviser. Nous souhaitons maintenant proposer une extension de ce modèle pour pouvoir utiliser des périphériques légers et en particulier des capteurs comme supports de CM et ainsi de les intégrer facilement dans les applications.

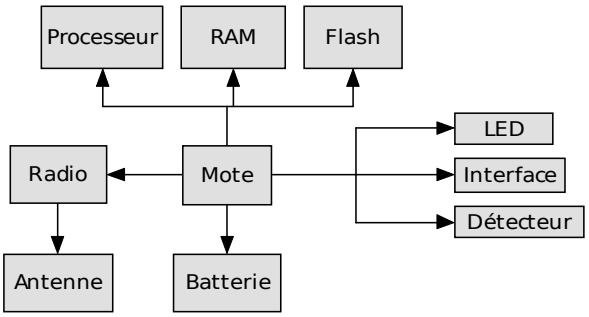

Figure 4 : *Architecture générale d'un capteur sans-fil*

3.1 *Modèle de capteur sans-fil*

Les récentes avancées dans le monde de la micro-électronique et des technologies sans-fil ont permis de développer des capteurs de petite taille, dotés de capacités de traitement et de modes de communication sans-fil. Certains permettent même des captures multimédia comme le son et l'image grâce à de petites caméras et des micros embarqués (Cyclops [14]).

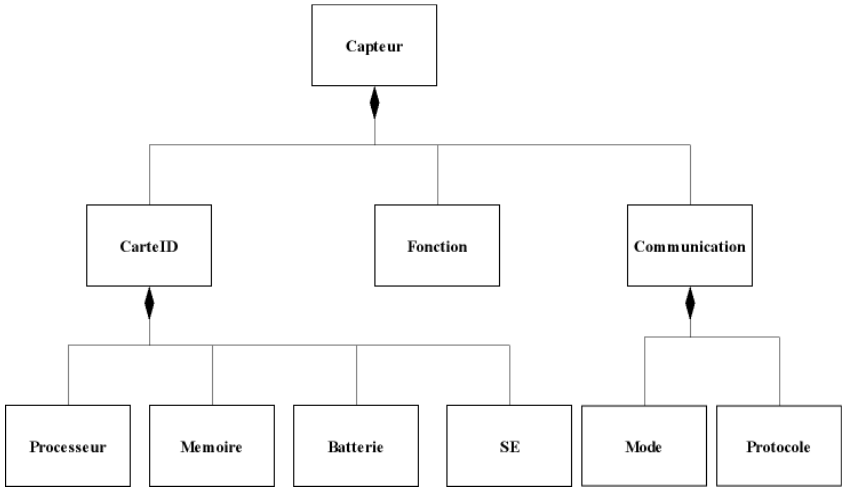

Figure 5 : *Diagramme de classe d'un capteur sans-fil*

Jusqu'à présent, aucun modèle commun à tous les capteurs sans-fil n'a été proposé. Dans l'objectif de leur intégration dans les applications, ce paragraphe propose une modélisation des capteurs sans fil que l'on peut trouver actuellement sur le marché.

Un capteur sans-fil est constitué d'un processeur, d'une mémoire, d'un émetteur-récepteur radio, d'une batterie et de détecteurs (Fig. 4) [1] [2] [3]. Il est composé de trois éléments : une carte d'identité, une ou plusieurs fonctions et un module de communication (Fig. 5). Sa carte d'identité est elle-même composée de quatre éléments : un processeur, une mémoire, une batterie et un système d'exploitation. Le module de communication est composé d'un mode, par exemple communication évènementielle, communication client/serveur, et d'un protocole de communication ou type de transmission comme le WIFI ou le Bluetooth. Le module de communication est doté d'un port permettant les entrées et sorties de messages et d'évènements.

Par exemple, la carte d'identité d'un capteur MICA2 de Crossbow est composée d'un processeur Atmega128 à 4MHz, d'une mémoire de 512 Ko pour stocker les mesures effectuées, d'une mémoire système de 128 Ko, d'une batterie composée de deux piles AA et du système d'exploitation TinyOS. Il communique de façon évènementielle et transmet messages et évènements par radio. Un capteur peut avoir plusieurs fonctions en intégrant plusieurs détecteurs. Il peut mesurer la température, la pression atmosphérique, l'humidité, le champ magnétique, la luminosité, le déplacement, capturer des sons et des images, etc. Lorsqu'il sera intégré dans une application, ce capteur jouera un rôle précis. Ce rôle fait appel à une, plusieurs ou toutes les fonctions du capteur. Le paragraphe suivant décrit l'intégration des capteurs sans-fil dans le modèle de composant unifié.

3.2 *Modèle unifié de composant*

Partant du constat que les programmes exploitant les fonctions des capteurs peuvent être considérés comme des composants métier, nous décrivons le modèle unifié de composant proposé pour modéliser des applications sans se soucier de la nature matérielle ou logicielle des composants qui les constituent. Ce modèle constitue une adaptation pour les capteurs sans-fil du modèle Osagaia [4].

Les capteurs sont capables de produire diverses sortes de flux. Pour traiter les informations qu'ils produisent, ils communiquent avec les composants logiciels procédant aux traitements spécifiques de ces informations. Pour que l'intégration de ces capteurs parmi les composants logiciels soit réalisable, nous proposons un modèle de composant unifié. Ce modèle propose d'intégrer l'exploitation des fonctions d'un capteur dans un PE du modèle Osagaia (Fig. 6). Le PE encapsule ces fonctions de la même façon qu'il le fait pour un composant logiciel. Le modèle proposé ne s'intéresse au capteur qu'au travers de ses fonctions. Les deux autres éléments constitutifs du capteur (carte d'identité et module de communication) relèvent de l'infrastructure matérielle de l'application. Ils seront pris en considération par la plate-forme de supervision en tant que supports potentiels de PE et de Conduits.

Dans le modèle Osagaia l'interconnexion des composants par les flux se fait grâce à la présence d'une UE et d'une US. De plus la plate-forme d'exécution supervise le CM au moyen d'une UC placée à l'intérieur du container (le PE). Afin de pouvoir l'interconnecter et le contrôler nous ajoutons au capteur, conformément au modèle Osagaia, une UC, une UE et une US. L'UC permet d'envoyer des commandes au CM et aux unités d'entrée et de sortie ainsi que de récupérer leurs états. Cette UC est capable de communiquer avec la mémoire et la carte d'identité du capteur pour rendre compte à tout moment de l'espace mémoire disponible et du niveau d'énergie mais également pour pouvoir superviser le capteur via son système d'exploitation. Ces différents ajouts se font par instanciation du modèle Osagaia. Ceci permet de réutiliser les composants déjà développés, faisant du capteur doté de ces unités, un cas particulier du modèle Osagaia.

Dans le modèle Osagaia, la plate-forme de supervision est distribuée sur tous les sites. En raison des contraintes de mémoire et des limites de capacité de calcul, il n'est pas possible d'héberger sur chaque capteur une partie de la plate-forme comme c'est fait sur chaque ordinateur. C'est pourquoi nous avons choisi de déporter la plate-forme ainsi que les UCs de chacun des composants hébergés sur le capteur vers le site le plus proche capable de les supporter.

Un capteur communique par sa radio (carte réseau sans-fil, etc) et c'est son seul moyen d'interaction avec les autres composants. Par conséquent, tous les échanges d'information se font par le port d'entrée/sortie du module radio (Fig. 6). Nous avons donc besoin de distinguer les données des flux d'états et de contrôle pour pouvoir les rediriger, en fonction de leur nature, vers l'entité correspondante (UE, US, UC).

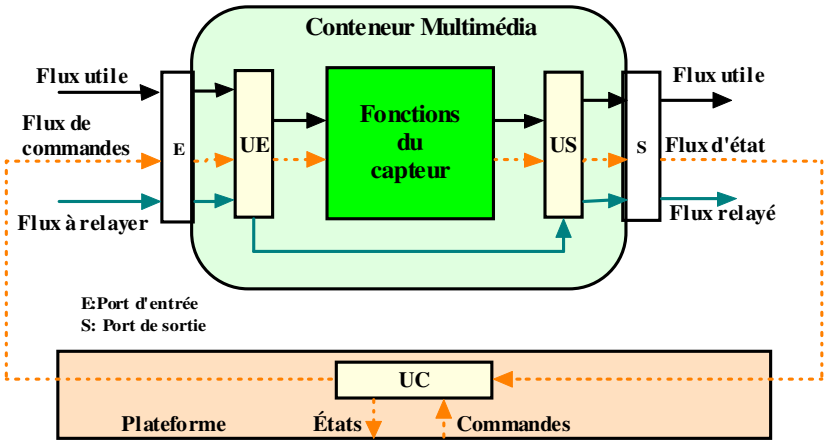

Figure 6 : *Intégration d'un capteur dans un PE OSAGAIA*

Pour cela nous utiliserons un modèle de flux dans lequel, aux informations (données, commandes, état), vient s'ajouter un identifiant qui permet d'indiquer si le flux est :

- un flux de données;
- un flux d'état;
- un flux de commandes.

La figure 7 montre une application composée d'un capteur vidéo mobile V, d'un capteur de son mobile S et d'un composant de mixage M situé sur une station fixe. Au dessous, un agrandissement de la station supportant M montre qu'une partie de la plate-forme s'y trouve ainsi que les UC des CM situés sur les capteurs V et S et le PE contenant le composant métier M. Le capteur V envoie un flux vidéo vers M, mais celui-ci étant hors de porté, S sert de relais. S reçoit ce flux, son UE l'identifie comme un flux à relayer et le dirige directement vers son US pour le transmettre. S transmet également le flux de son qu'il produit vers M. M lit les deux flux reçus dans son UE, les identifie comme flux utiles et les dirige vers son CM. De la même manière, lorsque la plate-forme doit envoyer un flux de commande vers V, elle le dirige vers S, qui le relaie vers V. Il en va de même lorsque V envoie des flux d'état vers la plate-forme. Cette fonction de relais permet de faire face à la mobilité des capteurs. Toutefois, afin de ne pas inutilement surcharger les capteurs par des fonctions de relais, la plate-forme étant répartie sur toutes les stations fixes, les UC des composants hébergés par les capteurs seront déplacées sur une station fixe directement à portée du capteur ou, à défaut, sur une station fixe nécessitant le moins de relais possible.

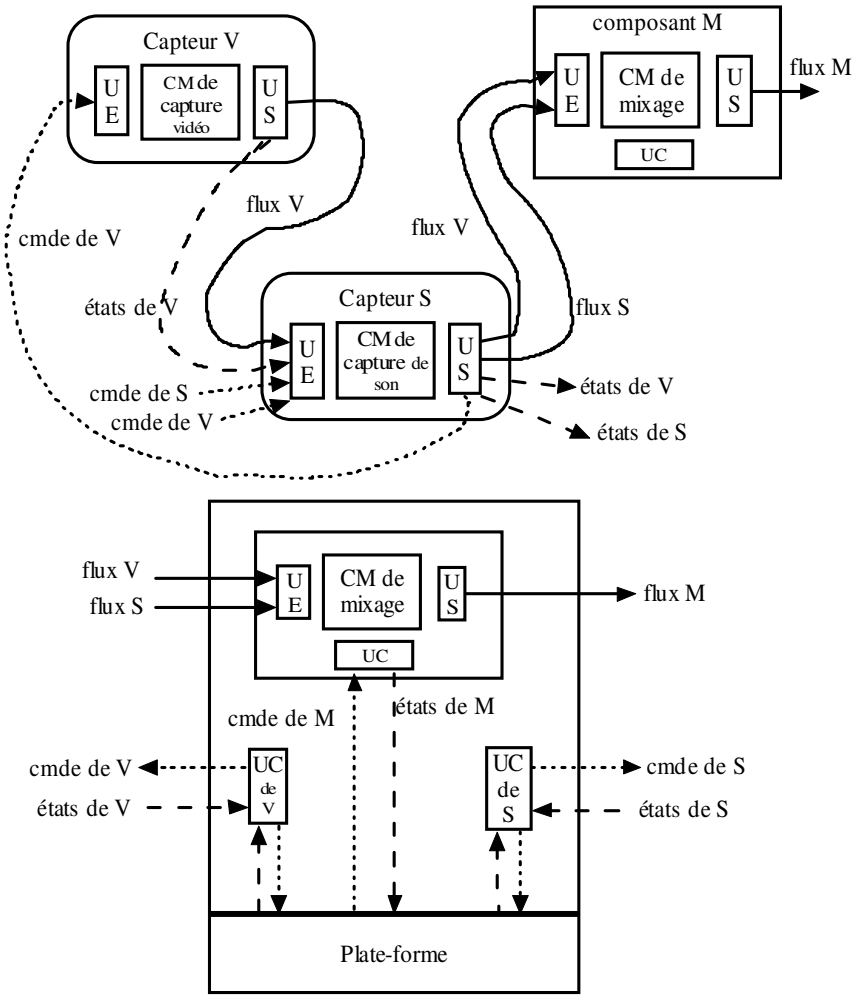

Figure 7 : *Flux dans une application composée de composants fixes et mobiles*

4. La plate-forme Kalinahia

Lors de travaux antérieurs nous nous sommes intéressés à la gestion de la qualité de service dans les applications multimédia réparties. La solution que nous avons retenue est celle de la reconfiguration dynamique des applications à l'aide d'une

plate-forme de supervision. En effet, pour que des reconfigurations puissent être dynamiquement effectuées, il est nécessaire que l'application soit réflexive dans le sens où elle doit en permanence avoir connaissance de sa structure et de son état. Afin de séparer les aspects fonctionnels des aspects non fonctionnels nous avons choisi de constituer la partie applicative à partir de composants interconnectés tandis qu'une plate-forme se charge du déploiement et de la reconfiguration. L'architecture générale se divise en deux parties décrites sur la figure 8 :

- Une partie applicative implémentant les fonctionnalités,
- Une plate-forme d'exécution et de supervision gérant dynamiquement la reconfiguration de la partie applicative.

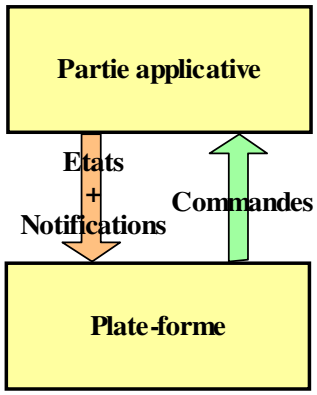

Figure 8 : *Architecture Kalinahia*

La plate-forme supervise et gère les composants formant la partie applicative tandis que ces derniers lui fournissent des états et des notifications sur leur fonctionnement. Ces informations offrent à la plate-forme une vision globale lui permettant de reconfigurer dynamiquement l'architecture de la partie applicative (par exemple en déplaçant un composant ou en le remplaçant par un ou plusieurs autres).

La plate-forme que nous proposons réalise une adaptation de l'application au contexte. Cette adaptation est entièrement dynamique puisque les politiques d'adaptation sont déterminées en cours d'exécution. De plus, elle concerne à la fois la structure, les fonctionnalités et l'ordonnancement des applications ce qui la rend plus complète que ce qui est souvent proposé — JQoS, Agilos [11] [17]. Ainsi, à partir de sa connaissance du contexte, des vœux des utilisateurs et des états fournis par la partie applicative, la plate-forme procède à un diagnostic de qualité de service et propose une nouvelle organisation susceptible de l'améliorer (Fig. 9). Elle utilise, pour ce faire, une heuristique pour trouver une solution satisfaisante à ce problème dont il est prouvé qu'il est NP-complet [10].

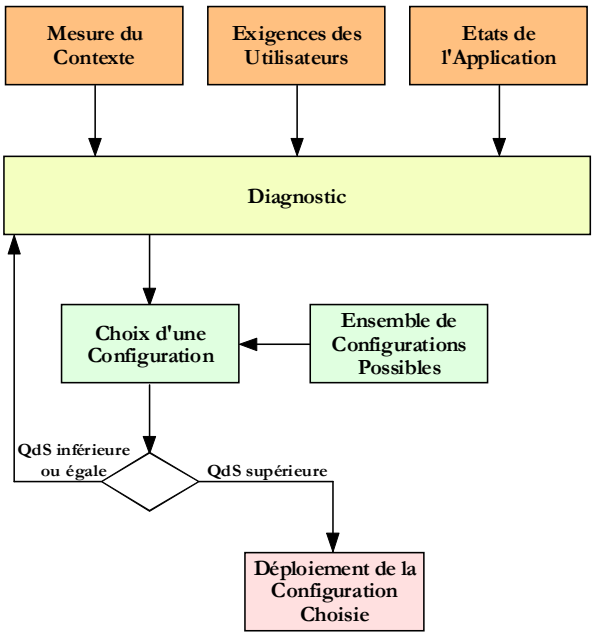

Figure 9 : *Fonctionnement de la plate-forme de reconfiguration*

5. Architecture des applications intégrant des capteurs

Les travaux décrits dans le paragraphe précédent et, plus précisément, dans [4] proposent un modèle de composant pour la reconfiguration dynamique d'applications multimédias distribuées. Ce modèle met en avant la séparation des préoccupations. En effet, il sépare la partie métier du composant (partie fonctionnelle) de la partie qui gère les propriétés non-fonctionnelles comme les échanges de données par exemple. Nous avons fait le choix de modéliser tous les composants de la même façon : une fonctionnalité -la partie métier- encapsulée dans un gestionnaire -un conteneur- constituant la partie non-fonctionnelle. Nous avons adopté ce modèle pour concevoir les applications constituées de composants logiciels et de capteurs. Nous l'avons adapté aux capteurs de façon à faire abstraction de la nature des composants et permettre une conception unifiée et simplifiée (Section 4)[12].

Le paradigme orienté service est aujourd'hui fréquemment utilisé pour le développement d'applications distribuées devant offrir une forte interopérabilité et évolutivité. Tout comme le paradigme composant, il favorise la réutilisation et la maintenance des applications mais il offre également des mécanismes pour la

description, la publication et la découverte. Un service est une entité autonome et indépendante de la plate-forme qui le supporte ce qui permet des reconfigurations aisées dans un environnement hétérogène. Enfin le mode de communication qu'ils utilisent permet une plus grande transparence. En effet, lorsque deux composants doivent dialoguer, il est nécessaire qu'ils se connaissent avant l'exécution alors que les services se lient à la volée, pendant l'exécution grâce aux mécanismes de découverte et de publication [5] [13]. Pour toutes ces raisons, et dans le but de gérer la qualité de service des applications distribuées dans des environnements hétérogènes, nous avons choisi de proposer une architecture à base de services.

Nous allons maintenant nous intéresser à l'évolution de la plate-forme permettant la supervision de telles applications.

5.1. *Evolution de la plate-forme Kalinahia*

Les architectures à services sont souvent utilisées pour les systèmes ayant besoin d'être dynamiquement reconfigurés. En effet leur principal atout est que les connexions entre les services sont complètement transparentes. Nos systèmes étant appelés, eux aussi, à être reconfigurés fréquemment pour garantir la meilleure qualité de service possible, nous avons fait le choix de concevoir une architecture à services. Elle est composée de quatre services : *Supervision*, *Usine à Conteneur*, *Usine à Conduit* et *Routage*.

Le service *Supervision* est le service principal de la plate-forme. Son rôle est de surveiller le fonctionnement de l'application. Pour cela il reçoit des différents composants de l'application des informations d'état à partir desquelles il évalue la qualité du service rendu. A l'issue de cette évaluation, il décidera de reconfigurer ou non l'application. Il a la charge d'envoyer aux services *Usine à Conteneur* et *Usine à Conduit*, qui seront décrits plus loin, les directives pour qu'ils créent les conteneurs et les composants de liaison adaptés à la nouvelle configuration et offrant le niveau de qualité de service requis. Pour évaluer au mieux la qualité de service et décider de la meilleure configuration possible, le service *Supervision* doit savoir quels sont les composants qu'il peut atteindre et, par conséquent, quelles sont les routes valides pour les atteindre. Il peut obtenir ces renseignements auprès du service *Routage* qui sera également décrit plus loin.

Le service *Usine à Conteneur* est un service permettant de construire un conteneur adapté au composant métier qu'il doit encapsuler. En effet, le conteneur n'est pas le même s'il doit encapsuler un composant logiciel situé sur un site non contraint, qu'on appellera site fixe (site dont les ressources ne sont pas critiques, par exemple un ordinateur de bureau) ou sur un site léger (site dont les ressources sont restreintes comme un capteur sans-fil, un téléphone mobile, un PDA, etc.) Le conteneur est également différent selon que le composant métier qu'il encapsule utilise un mécanisme de communication par évènement ou un mécanisme de communication par appel de méthode ou encore par boîte à lettres. Les CM disponibles sont situés dans un entrepôt. Le service *Usine à Conteneur* doit donc se connecter à l'entrepôt

pour charger le CM demandé par le service *Supervision*. Ensuite il adapte l'interface de communication au mécanisme utilisé par le CM et déploie le CM et le PE adapté. Lorsque survient une reconfiguration, le service *Supervision* lui indique le CM et par conséquent le conteneur qu'il doit supprimer.

Le service *Usine à Conduit* est un service permettant de construire tout type de conduit adapté à l'application : transmission synchrone des données ou non, transmission en temps réel ou non, etc. Lorsqu'une reconfiguration est requise, ce service est averti par le service *Supervision* de la nécessité de relier des composants métier par un conduit. Il obtient de la part du service *Supervision*, la localisation du composant source et celle du composant cible. Il obtient également les informations de contraintes de transmission liées à l'application : synchrone, asynchrone, temps réel ou non, etc. A partir de ces informations, il construit le conduit adapté et connecte chacune de ses extrémités aux composants à relier. Les informations de reconfiguration envoyées par le service *Supervision* peuvent également provoquer des changements de points de connexion. Ces changements seront effectués par la suppression du conduit jusqu'alors utilisé et la construction d'un nouveau conduit de source et/ou de cible différente.

Le service *Routage* est un service permettant de conserver une table de routes à jour pour atteindre tous les composants de l'application. Il est à la fois un service espion et un service d'information. Il est un service espion lorsqu'il détecte la disparition d'une route en cours d'utilisation. Dans ce cas, il avertit par un message prioritaire le service *Supervision* qu'il est nécessaire et urgent d'effectuer une reconfiguration pour trouver une autre route et continuer d'assurer le fonctionnement de l'application. Il assure également un service d'information lorsqu'il détecte un changement de route, en avertissant le service *Supervision* qui décidera, selon la route concernée, d'évaluer s'il est opportun de reconfigurer l'application.Les applications utilisent différents types de dispositifs pour supporter leurs composants (mobile, non mobile, ressources contraintes ou non, etc.). Par conséquent deux catégories d'hôtes (ou sites) ont été définies. Les capteurs et les périphériques mobiles sont qualifiés d'hôtes légers en raison de leurs contraintes de ressources. A l'inverse d'un hôte fixe comme un PC, un hôte léger a de fortes contraintes d'énergie, de fortes limitations de capacité de calcul et de réseau, etc. Les paragraphes suivants décrivent, pour chaque catégorie d'hôte, la démarche de déploiement des composants.

5.2. *Cas d'un hôte fixe*

La plate-forme de supervision, au même titre que l'application, est distribuée sur tous les hôtes participant à la configuration. Chaque hôte héberge une partie de la plate-forme mais tous les hôtes ne peuvent pas supporter l'ensemble des services. Nous considérons les hôtes fixes, tels les PCs, comme ayant des contraintes de ressources négligeables face à celles des hôtes de plus en plus légers comme les PDA, les téléphones mobiles, les capteurs sans fils, etc. Les hôtes fixes sont capables

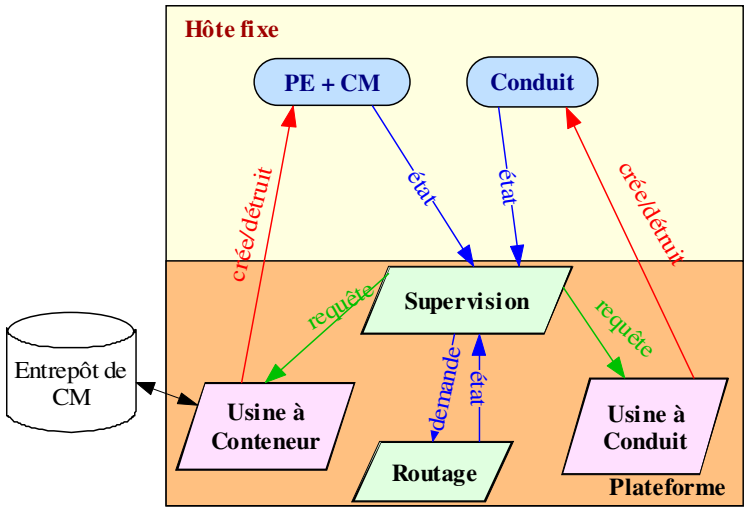

Figure 10 : *Déploiement de la plate-forme sur un hôte fixe*

d'héberger les quatre services précédemment décrits et de les faire fonctionner (Fig. 10). Pour les mêmes raisons, les conteneurs et les conduits peuvent être déployés dans leur totalité sur un hôte fixe. Il n'en va évidemment pas de même pour les hôtes légers.

Le service *Supervision* avec l'aide du service *Routage* a connaissance à tout instant de la configuration de l'application. A chaque reconfiguration, il envoie les informations nécessaires aux services d'*Usine à Conteneur* et d'*Usine à Conduit* pour créer et/ou supprimer des PEs et des conduits.

Quand il déploie un PE, le service *Usine à Conteneur* le déploie avec ses unités d'échange et son UC. De la même façon, le service *Usine à Conduit*, déploie sur l'hôte fixe source l'extrémité du conduit contenant le port d'entrée avec une partie de l'UC et sur l'hôte cible l'extrémité contenant le port de sortie avec une partie de l'UC (ou tout sur le même hôte s'il s'agit d'une connexion interne). Dans le paragraphe suivant, nous verrons que le déploiement des PEs et des Conduits est différent pour un hôte léger.

5.3. *Cas d'un hôte léger*

Un hôte léger est limité en termes d'énergie, de capacité de calcul et de réseau. Par conséquent, il n'est pas souhaitable de le surcharger en y hébergeant la totalité des services de la plate-forme comme on le fait un hôte fixe. Une sélection des services indispensables a été faite. Ce sont ces services qui seront effectivement hébergés sur l'hôte léger. Les autres seront déportés sur l'hôte fixe le plus proche. Ainsi le service

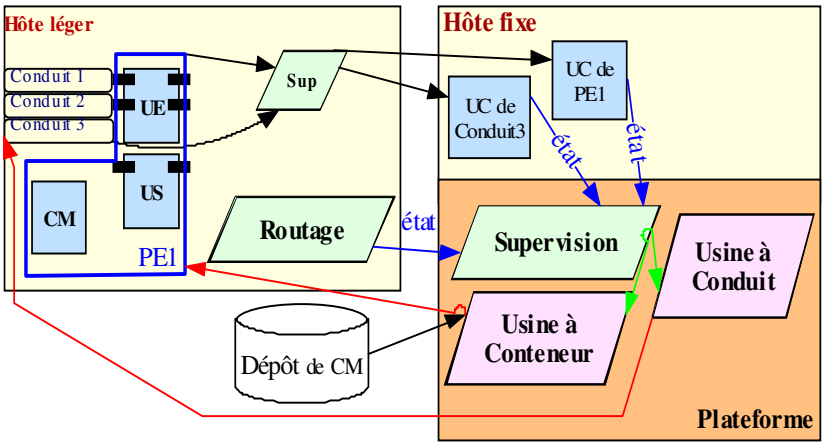

Figure 11 : *Déploiement de la plate-forme sur un hôte léger.*

Routage doit impérativement se trouver sur l'hôte léger sinon il n'a aucun moyen de connaître tous les hôtes (et par conséquent tous les composants) qu'il peut atteindre. En revanche, la supervision des hôtes légers sera assurée par le service *Supervision* de l'hôte fixe le plus proche, appelé hôte correspondant (Fig. 11). Les services *Usine à Conteneur* et *Usine à Conduit* sont pilotés par le service *Supervision*, ils seront donc de la même façon situé sur l'hôte correspondant.

La démarche de déploiement des PE et des Conduits est également différente de celle utilisée pour les hôtes fixes. Pour éviter de surcharger les hôtes légers et pour minimiser les transmissions réseau qui consomment beaucoup d'énergie sur les capteurs, les UC des PE sont déportées sur l'hôte correspondant. L'UC n'est sollicitée que ponctuellement par le service *Supervision* contrairement aux unités d'échange indispensables pour les communications avec les autres composants de l'application. C'est pourquoi, déporter l'UC sur l'hôte correspondant n'aura pas de lourdes conséquences sur la consommation d'énergie des hôtes légers mais permettra de libérer des ressources de mémoire et de processeur. De la même façon, les UC des conduits sont déportées sur l'hôte correspondant. La table 1 récapitule le déploiement des conduits et des PE encapsulant les composants métier. Les parties indispensables sont supportées par l'hôte léger et les autres par l'hôte correspondant.

Cependant les hôtes légers tels les capteurs, les PDA et les téléphones mobiles peuvent se déplacer et ainsi perdre la liaison réseau avec leur hôte correspondant. Dans ce cas, le service *Routage* alertera le service *Supervision* que la topologie a été modifiée. Ce dernier va rechercher un nouvel hôte correspondant et ordonner d'y déplacer les UC orphelines. Nous retrouvons un mécanisme comparable de migration pour les capteurs sans-fil dans [16].

	Conduit	PE + CM
Hôte fixe	Extrémité (UE ou US) + UC	CM + UE + US + UC

	Conduit	PE + CM
Hôte léger	Extrémité (UE ou US)	CM + UE + US
Hôte correspondant	UC	UC

Table 1. *Déploiement de composants sur un hôte fixe et sur un hôte léger*

Dans la pratique, les capteurs font partie d'une catégorie particuliere d'hôte léger, très contraint, que nous comparerons au standard CLDC[1] de la machine virtuelle Java. Les capteurs ont la particularité de ne pas contenir les outils nécessaires pour le chargement dynamique de composant. En effet, les différents programmes que l'utilisateur souhaite exécuter sur un capteur doivent être chargés sous forme de paquetages. Chaque fois qu'un composant doit être ajouté, il faut créer un nouveau paquetage comportant les composants actuels ainsi que le nouveau composant et ensuite le charger sur le capteur.

Nous proposons donc un déploiement spécifique pour les capteurs. En effet, puisque ni les conteneurs, ni les conduits ne peuvent être déployés à la volée, nous proposons que tous les services de la plateforme soient hébergés sur le capteur. Cependant, le service *Usine à Conteneur* est réduit à ne produire que des conteneurs spécifiques au capteur. De la même façon, le service *Usine à Conduit* ne produit que des Conduits spécifiques aux contraintes du capteur. Enfin, le paquetage chargé sur le capteur comprendra également le dépôt des CM qui pourront être mis en œuvre.

6. Implémentation

Le modèle de composants unifié Osagaia a été implémenté en Java tel que nous l'avons présenté dans la section 3. Cette implémentation a donné lieu à l'élaboration d'un prototype de vidéoconférence permettant une adaptation dynamique au contexte et mettant en œuvre des conduits proposant une politique de synchronisation des flux de données. L'archive, implémentée en Java, est disponible à l'url suivante : http://www.iutbayonne.univ-pau.fr/~roose/pub/recherche/Osagaiakorrontea.

La plate-forme de reconfiguration, quant à elle (Fig. 2), a fait l'objet de la réalisation d'un simulateur sous Labview disponible à l'url suivante :

[1] Connected Limited Device Configuration

http://www.iutbayonne.univ-pau.fr/~roose/pub/recherche/kalinahia. Ce simulateur permet de montrer que la plate-forme, après plusieurs choix de configuration, converge vers une configuration stable de l'application permettant d'obtenir une QdS tendant vers l'optimum (ce dernier ne pouvant être atteint a priori puisqu'il s'agit d'un problème NP-complet) [10].

Les différentes implémentations précédemment citées, bien qu'elles soient complètes d'un point de vue fonctionnel, ne peuvent pas être déployées telles quelles sur des capteurs, beaucoup trop contraints matériellement. Nous travaillons actuellement à l'implémentation de la plate-forme particulièrement allégée pour les périphériques fortement contraints. Notre choix de capteurs s'est porté sur des Sun Spot de Sun Microsystems. Ce choix est guidé par leur capacité à implémenter une machine virtuelle Java compatible CLDC 1.1, Squawk [16].

Il est à noter que nos travaux sont en partie financés par Sun Microsystems avec qui nous sommes en partenariat.

7. Travaux Apparentés

Bon nombre de travaux portent sur les architectures logicielles de reconfiguration des réseaux de capteurs et sur la collaboration entre composants logiciels et matériels.

Les auteurs de [15] proposent une architecture pour configurer des réseaux de capteurs de façon rapide et dynamique. C'est une combinaison d'une architecture matérielle et d'une architecture logicielle qui permet l'échange d'information entre un réseau de capteur et d'autres applications. Un serveur gère les informations recueillies par les capteurs et les transmet vers les applications. Jusqu'à présent, lorsqu'une nouvelle fonctionnalité devait être chargée sur un capteur, il fallait mettre à jour le serveur pour qu'il charge le code nécessaire pour réaliser la translation des données brutes en données interprétables par les applications. [15] propose de détecter automatiquement la présence de nouveaux capteurs et d'en télécharger le composant logiciel permettant la translation des données. Ainsi il est possible de développer un serveur générique et réutilisable qui ne nécessite pas de mise à jour à chaque changement de fonctionnalité. Pour cela ils utilisent la technologie JavaBean pour décrire le réseau de capteur. Chaque capteur renferme deux JavaBeans : l'un contient les données spécifiques au capteur (Identifiant, localisation, type, etc.) et l'autre contient les méthodes pour accéder au capteur (translation des données, etc.). Le code occupe très peu de place dans la mémoire des capteurs, cependant la transmission d'un JavaBean peut demander une forte consommation d'énergie et réduire considérablement la durée de vie du capteur. Une alternative possible est de ne transmettre qu'une URL où le serveur peut ensuite télécharger le composant. Pour exporter les fonctions des capteurs et les rendre utilisables par les applications, un framework OSGi est installé sur le serveur. Ce framework offre une gestion aisée de la mobilité des capteurs par son mécanisme d'ajout et suppression de services.

L'architecture proposée dans [15] est intéressante par la flexibilité que procure le découplage entre les données spécifiques du capteur et son comportement. Dans nos recherches, nous avons également fait de choix de séparer deux aspects des composants : une partie fonctionnelle qui concerne le traitement des données et une partie non-fonctionnelle qui traite du transport de l'information et du contrôle de la partie fonctionnelle. Dans le même esprit de conservation de l'énergie, nous déportons des services de la plate-forme et des parties des composants sur des hôtes moins contraints. Enfin, nous avons fait le choix d'une plate-forme distribuée plutôt qu'un serveur centralisant les informations des capteurs et les transmettant aux applications. En effet l'évaluation de la qualité de service et le choix d'une nouvelle configuration devant être faits en temps réel, seule une méthode de calcul réparti peut y parvenir [9]. De plus, nous considérons les capteurs comme faisant partie intégrante des applications au même titre que les composants logiciels. Les capteurs et les composants logiciels collaborent directement pour réaliser le service rendu par l'application.

Les auteurs de [6] proposent un intergiciel évolutif selon les changements de l'environnement. Les capteurs doivent pouvoir adapter leur comportement en fonction des conditions environnementales. Ils doivent pouvoir également adapter leur mécanisme de communication ou encore leur protocole réseau. La reconfiguration des réseaux de capteurs est décrite selon deux dimensions : une dimension locale, au niveau du fonctionnement du capteur et une dimension distribuée, au niveau du fonctionnement du réseau de capteurs. Ils proposent deux architectures. La première est une architecture locale aux capteurs. Chaque capteur est un composant composite où les connexions entre les composants internes sont contrôlées par un ensemble de règles ordonnées par un configurateur. Lorsque survient un changement de contexte, le configurateur va engendrer une reconfiguration en ajoutant ou supprimant des composants internes et/ou des connexions. La deuxième architecture regroupe un ensemble d'architectures locales de même type, c'est-à-dire réalisant toutes la même fonctionnalité. Une reconfiguration au niveau réseau engendre une reconfiguration des architectures locales aux capteurs.

Tous les composants du réseau offrent la même fonctionnalité alors que nos applications sont le résultat de la collaboration de fonctionnalités différentes. L'architecture distribuée est similaire à notre plate-forme distribuée sauf qu'elle ne se préoccupe que du fonctionnement du réseau et du calibrage des capteurs mais ne s'intéresse pas à l'ajout, la suppression et la modification des fonctionnalités des capteurs.

L'architecture proposée par [7] [8] permet de modifier la fonctionnalité de chaque nœud du réseau de capteur en réponse à un changement de contexte environnemental ou applicatif. Chaque nœud renferme un composant *monitor*, un composant *configurator* et sa fonctionnalité. Les capteurs sont reliés à une base où sont hébergées les différentes configurations possibles, les exigences de QdS et le code à charger sur les nœuds correspondant à chaque configuration. Le composant m*onitor* mesure la QdS locale au capteur et la transmet à la base qui l'évalue. Il joue un rôle

comparable à l'Unité de Contrôle des composants de notre architecture. Il faut toutefois remarquer que dans notre modèle, les PEs et les Conduits ayant une UC, notre évaluation de la QdS porte à la fois sur les traitements et sur les circulations d'informations. Le composant d'évaluation de la QdS de la base joue un rôle comparable au service *Supervision* que nous proposons. Le composant *reconfigurator* a la charge de reconfigurer la fonctionnalité locale au capteur sur ordre de la base. Il joue un rôle comparable au regroupement des services *Usine à Conteneur* et *Usine à Conduit* de notre architecture. Pour des raisons de conservation de l'énergie des capteurs, nous avons fait le choix de déporter les services *Usine à Conteneur* et *Usine à Conduit* ainsi que les UC des composants. De même, les reconfigurations dans [7] sont effectuées selon des configurations préétablies comme il est fait dans [9]. Notre architecture ne se contente pas de reconfigurer les réseaux de capteurs, elle permet de reconfigurer des applications hétérogènes constituées de composants de nature différentes tels que les capteurs et les composants logiciels. De plus, elle permet de prendre en compte des contraintes liées au traitement de données multimédias comme la synchronisation.

8. Conclusions et perspectives

Les capteurs deviennent de plus en plus présents autour de nous. Ils sont dotés de capacités de traitement et d'une mémoire et peuvent aussi bien mesurer des températures que capturer des sons ou des images. Notre objectif est de les intégrer dans des applications à la fois en tant que pourvoyeurs d'information et que supports de composants logiciels. Pour concevoir de façon aisée ce genre d'applications, nous proposons un modèle unifié de composants qui permet au développeur de ne pas se préoccuper de la nature physique ou logicielle des entités. Dans cet article nous avons présenté le modèle de composants Osagaia et nous avons vu que l'on pouvait l'adapter aussi bien aux composants logiciels supportés par des machines classiques qu'à ceux supportés par des capteurs. Toutefois nous avons dû le modifier de façon à prendre en compte la faible capacité des capteurs et leur mobilité. En outre, nous proposons d'utiliser les unités d'entrée et de sortie du modèle pour mettre en place de façon simple une fonction de relais pour palier au problème de portée des capteurs.

Ce modèle unique ouvre la voie à la conception d'applications par interconnexion de composants matériels et logiciels sans qu'il soit besoin d'adaptation particulière en fonction des composants utilisés. La plate-forme dispose alors de tous les moyens lui permettant de superviser ces composants en réorganisant la circulation des flux entre eux. Elle reçoit de chacun d'entre eux des états à partir desquels elle détermine comment l'application s'exécute et leur envoie des commandes pour en piloter le fonctionnement.

Jusqu'à présent, la plupart des applications utilisant des capteurs séparent le réseau de capteurs et les logiciels utilisant leurs informations. Nous proposons de dépasser cette distinction et de considérer les capteurs comme faisant partie intégrante des applications en proposant une architecture distribuée dynamiquement reconfigurable

à laquelle ils participent pleinement. De plus, les capteurs sont considérés comme de potentiels supports pour des composants logiciels. Ceci permet de proposer des organisations dans lesquelles les transferts d'informations qui sont de gros consommateurs d'énergie peuvent être minimisés en plaçant les composants de traitement sur ou au plus près de ceux qui produisent les informations à traiter. Ainsi, sur l'exemple de l'application de télésurveillance, le fait de pouvoir placer un composant de détection de mouvement sur un capteur vidéo utilisé en tant que détecteur de présence évite d'avoir à transmettre cette vidéo. De la même façon la possibilité d'effectuer un traitement sur un capteur servant de relais de transmission d'information peut permettre de ne transmettre que les informations strictement nécessaires.

L'architecture logicielle que nous proposons est constituée de quatre services : *Supervision*, *Usine à Conteneur*, *Usine à Conduit* et *Routage*. Le service *Supervision* surveille le niveau de QdS de l'application et réagit aux changements du contexte. En relation avec le service *Routage*, il contrôle les services *Usine à Conteneur* et *Usine à Conduit* pour ajouter et/ou supprimer des composants et des connexions adaptés aux besoins de l'application.

Peu de travaux sur les architectures pour les réseaux de capteurs relèvent le défit du déploiement et de l'hétérogénéité. Les applications sont, le plus souvent, constituées de dispositifs sensiblement identiques. Les applications visées par nos travaux sont constituées de composants aux contraintes différentes désignés comme hôtes légers et hôte fixes. Les composants et les conduits sont déployés différemment en fonction des possibilités de l'hôte. Nous répondons aux contraintes d'énergie et de mémoire des hôtes légers en déportant une partie des composants et des services de la plate-forme sur un hôte fixe voisin. Le service *Routage* permet de faire face à la mobilité des hôtes légers et de déplacer les services déportés vers l'hôte fixe le plus proche.

Les travaux à court terme concernent le développement et le déploiement de l'architecture proposée sur un système composé de PCs et de capteurs Sun Spots. Ils concerneront particulièrement le choix des composants contenus dans les paquetages déployés sur les capteurs. En effet ce paquetage doit être conçu en tenant compte des contraintes de mémoire, d'énergie et de mobilité. Le paquetage doit être le plus complet et pertinent possible afin de limiter la fréquence de changement de paquetage, chaque chargement entrainant une forte consommation d'énergie et une perte de mesure du contexte. A moyen terme, ils concernent la conception d'un système d'information unifié correspondant au fonctionnement de l'architecture c'est-à-dire à la description de tous les messages émis et reçus par la plate-forme : création et destruction de conteneurs et de conduits, demande d'état de fonctionnement des composants et des conduits et demande d'état des routes.

9. Bibliographie

[1] Akyildiz I. F., Su W., Sankarasubramniam Y., « Wireless Sensor Networks: a survey', *Computer Networks*, Volume 38, No. 4, pp 393-422, 2002.

[2] Blumenthal J., Handy M., Golatowski F., Haase M., Timmermann D., « Wireless Sensor Networks – New Challenges in Software Engineering », *IEEE Conference ETFA'03*, Volume 1, pp. 551-556, 2003.

[3] Culler D., Estrin D., Srivastava M., « Overview of Sensor Networks », *IEEE Computer*, August 2004, pp 41-49.

[4] Dalmau M., Roose P., Bouix E., Luthon F., « A Multimedia Oriented Component Model », *The IEEE 19th International Conference on Advanced Information Networking et Applications, AINA 2005*, 2005.

[5] Elfatatry A., « Dealing with change: Components versus Services », *Communications of the ACM*, August 2007, Volume 50, No. 8.

[6] Grace P., Coulson G., Blair G., Porter B., and Hughues D., « Dynamic Reconfiguration in Sensor Middleware », *MidSens'06*, (Melbourne, Australia, November 27-December 1), 2006.

[7] Kogekar S., Neema S., and Koutsoukos X., « Dynamic Software Reconfiguration in Sensor Networks », *Systems Communications 2005 (ICW / ICHSN / ICMCS / SENET 2005)*, IEEE Computer Society, (14-17 August 2005, Montreal, Canada), 2005.

[8] Kogekar S., Neema S., Eames B., Koutsoukos X., Ledecz, A., and Maroti M., « Constraint-guided Dynamic Reconfiguration in Sensor Networks », *Proceedings of the Third International Symposium on Information Processing in Sensor Networks, IPSN 2004*, (Berkeley, California, USA, April 26-27), 2004.

[9] Laplace S., Dalmau M., Roose P., « Kalinahia: Considering quality of service to design and execute distributed multimedia applications », *IEEE/IFIP Int'l Conference on Network Management and Management Symposium*, *NOMS 2008*, (Salvador de Bahia, Brésil, Avril, 2008), 2008.

[10] Laplace S., Dalmau M., Roose P., « Kalinahia : modèle de qualité de service pour les applications multimédia reconfigurables »,- *Numéro Spécial Revue ISI « Conception : patrons et spécifications formelles",* Volume 4, 2007.

[11] Li B., Kalter W., Nahrstedt K., « A Hierarchical Quality of Service Control Architecture for Configurable Multimedia Applications », Journal of High Speed Networks, Special Issue on Management of Multimedia Networking, IOS Press, Vol. 9, pp. 153-174, 2001.

[12] Louberry C., Roose P., and Dalmau M., « Heterogeneous component interactions: Sensors integration into multimedia applications », *Journal of Networks*, Academy Publisher, 2007, Volume 3, No. 2.

[13] Papazoglou M. P., Traverso P., Dustdar S., and Leymann F., « Service-oriented computing: state of the art and research challenges », *IEEE Computer*, November 2007, Volume 40, No. 11.

[14] Rahimi M., Baer R., Iroezi O. I., Garcia C., Warrior J., Estrin D., Srivastava M., « Cyclops : In Situ Image Sensing and Interpretation in Wireless Sensor Network », *ACM SENSYS*, pp. 192-204, 2005.

[15] Russo J., Helal A., King J., and Bose R., Self-describing sensor networks using a surrogate architecture, Internal Report, Mobile&Pervasive computing research, University of Florida, June, 2005.

[16] Smith R. B., Cifuentes C., and Simon D., « Enabling Java™ for small wireless devices with Squawk and Spotworld », *Workshop on Building software for pervasive computing, OOPSLA'05*, (San Diego, California), 2005.

[17] Zhu W., Georganas N. « JQoS : Design and Implementation of a QoS-based Internet Videoconferencing System using Java Media Framework (JMF) » CCECE'2001, Canadian Conference on Electrical and Computer Engineering, Toronto, Canada.