Management of Data Sources and Services in Pervasive Environments

Yann Gripay
Université de Lyon
INSA-Lyon
LIRIS UMR 5205 CNRS
Bat Blaise Pascal
69621 Villeurbanne Cx
+33/0 472 43 88 99
yann.gripay@liris.cnrs.fr

Frédérique Laforest
Université de Lyon
INSA-Lyon
LIRIS UMR 5205 CNRS
Bat Blaise Pascal
69621 Villeurbanne Cx
+33/0 472 43 88 97
frederique.laforest@liris.cnrs.fr

Jean-Marc Petit
Université de Lyon
INSA-Lyon
LIRIS UMR 5205 CNRS
Bat Blaise Pascal
69621 Villeurbanne Cx
+33/0 472 43 79 24
jean-marc.petit@liris.cnrs.fr

Liya Zeng
Université de Lyon
INSA-Lyon
Dpt Télécommunications
Bat A. de St Exupery
69621 Villeurbanne Cx
liya.zeng@insa-lyon.fr

ABSTRACT

Our daily environment contains more and more computerized devices, and tends to become a pervasive environment. Pervasive applications allow the end user to use a large number of heterogeneous devices, potentially invisible to him/her. To manage such kinds of applications, a distributed and modular architecture is required to handle the devices heterogeneity and their dynamicity, concerning their availability as well as the nature of data and services they can provide. In this article, we propose an architecture integrated in our SoCQ project [16] that builds an abstraction of the environment and thus simplifies the development of pervasive applications based on this abstraction. We also detail the current implementation of our architecture that includes the use of the UPnP technology.

Categories and Subject Descriptors

C.2.4 Distributed systems, H.2.4. Systems for database management

General Terms

Design, Experimentation.

Keywords

Systèmes pervasifs, requêtes continues, flux de données, services, UPnP

1. INTRODUCTION ET MOTIVATION

Les environnements informatiques évoluent vers ce qui est appelé les systèmes pervasifs [22] : ces environnements sont de plus en plus hétérogènes, décentralisés et autonomes. D'une part, les PCs et autres terminaux mobiles sont maintenant courants et composent une grande partie des systèmes d'information. D'autre part, les ressources de l'environnement sont souvent réparties sur de larges zones à travers des réseaux qui peuvent aller d'un réseau mondial comme Internet jusqu'aux réseaux locaux pair-à-pair comme pour des capteurs autonomes.

NOTERE 2008, June 23-27, 2008, Lyon, France.

Les problématiques des systèmes pervasifs sont vastes. Dans notre projet SoCQ [14, 15, 16], nous nous focalisons sur les deux points suivants :

- Données, services, capteurs

Les environnements pervasifs incluent souvent des *bases de données* classiques et parfois réparties, mais aussi des *capteurs* fournissant des flux de données, et des *services* distants permettant d'effectuer des appels de méthodes ou d'agir sur l'environnement (*actionneurs*). Bases de données, capteurs et services sont répartis dans l'environnement. Le principe de dataspace [10, 17] repose sur l'idée que l'utilisateur est entouré d'un ensemble de ressources vues d'une façon uniforme, que ces ressources soient des bases de données, des capteurs ou des services. Dans cette optique, toutes ces ressources peuvent être vues comme des données, qu'elles soient persistantes ou calculées à la demande. Ainsi, la construction d'une application pervasive pourrait être exprimée à l'aide d'un langage déclaratif. En effet, présenter l'ensemble des ressources dans une forme unifiée relationnelle permettra à l'utilisateur concepteur d'applications de construire la partie fonctionnelle de son application sous la forme de requêtes à la SQL [14, 18, 24]. Le moteur de traitement de requêtes nécessaire dans cette vision doit intégrer dans les bases de données à la fois des tables relationnelles, des flux de données provenant de capteurs, des services fournisseurs de données et des services liés à des actionneurs qui agissent sur l'environnement.

- Environnement évolutif et adaptabilité

L'une des caractéristiques des systèmes pervasifs est l'évolutivité de l'environnement. Des ressources sont disponibles à certains moments, et peuvent apparaître ou disparaître à tout moment. L'un de nos objectifs est donc de prendre en compte cette évolutivité de l'environnement. L'évolution de l'environnement ne doit pas perturber l'utilisateur. Par exemple, la disparition d'un capteur ou l'apparition d'un actionneur ne doit pas être intrusive : le système pervasif doit s'adapter aux modifications de la façon la plus transparente possible.

Dans notre projet SoCQ, le framework offre une représentation homogène des bases de données relationnelles, des flux de données et des services, et permet de définir des requêtes continues combinant tout type de ressources. Les ressources sont représentées sous forme de tables (relations ou flux) comportant des attributs réels (des attributs classiques dans des tables de bases

relationnelles) et/ou des attributs virtuels, qui représentent les paramètres d'entrée/sortie des services. Les requêtes SoCQ, exprimées dans un langage à la SQL, permettent ainsi de définir une partie du comportement des applications pervasives combinant tout type de ressources. Cependant, le moteur de requêtes SoCQ dans cette version ne répond pas à la gestion adaptative et transparente de l'évolutivité de l'environnement. Nous avons donc défini une architecture intégrant le moteur de requêtes SoCQ et la gestion des ressources.

Dans cet article, nous présentons brièvement le moteur de requêtes SoCQ puis l'architecture permettant de l'augmenter pour qu'il prenne en compte de façon transparente l'évolutivité de l'environnement. Un prototype a été développé, et nous présentons des premiers résultats au travers d'un scénario.

2. TRAVAUX EXISTANTS

Les systèmes pervasifs ont été étudiés dans de nombreux travaux [3, 4, 8, 12, 13, 27 *etc.*]. Ils abordent le problème de la découverte dynamique [8, 27] par une abstraction du réseau physique et des fonctionnalités des équipements [3, 4]. Le partage des données et des applications entre des équipements répartis, notamment des équipements d'interface avec les utilisateurs, est simplifié [12]. D'une manière plus centrée sur l'utilisateur, [13] introduit la notion de tâche, c'est-à-dire l'ensemble des applications et des données qu'un utilisateur est en train d'utiliser, qui doit être instancié dans tout environnement où l'utilisateur se trouve. L'environnement de l'utilisateur peut devenir un « espace pervasif programmable » [17] où les applications seraient définies uniquement sous forme de composition de services. Nous avons pour notre part choisi d'étudier un autre mode de représentation de l'environnement, non pas basé sur la composition de services, mais sur un modèle relationnel des ressources et sur l'utilisation de la puissance d'expression de l'algèbre relationnelle. Ce mode de représentation par un modèle relationnel nous permet également d'intégrer les travaux sur les flux de données, utiles par exemple pour prendre en compte les données issues de capteurs ou tout autre système événementiel.

De très nombreux travaux ont été réalisés sur les requêtes continues. La plupart des articles (par exemple [2, 5, 11, 26]) proposent une extension de SQL pour travailler à la fois avec des bases de données relationnelles et des flux de données. Certains travaux aborde les requêtes continues sur des données XML réparties [6] ou en représentant les requêtes sous forme de flux de tuples à travers des « boîtes » (les opérateurs) [1, 7].

Cependant, à notre connaissance, cette notion de requête continue a eu peu d'impact sur le développement d'applications pervasives impliquant non seulement des bases de données classiques et des flux de données, mais aussi des services. Il existe quelques exemples cependant. Dans [25], les requêtes continues peuvent implicitement interagir avec un équipement à travers un appel de fonction externe. Dans [19], le processus de nettoyage de données issues de capteurs physiques est défini de manière déclarative par un *pipeline* de requêtes continues. Mais ces applications sont souvent développées de manière *ad hoc*. Dans le *framework* SoCQ [14, 15, 16], nous avons défini une représentation homogène pour les sources de données non conventionnelles afin de définir entièrement des applications pervasives de manière déclarative par un ensemble de requêtes continues orientées service.

3. ARCHITECTURE PERVASIVE DE GESTION DE RESSOURCES

3.1 Moteur de requêtes SoCQ

Le moteur de requêtes continues orientées service, ou *SoCQ Processor* (Service-oriented Continuous Query Processor), est un système de gestion de bases de données, de flux de données et de services.

D'une part, c'est un système de gestion de base de données et de flux de données : il permet d'exécuter des requêtes continues combinant ces deux types de données. En plus des opérateurs classiques du SQL, qui travaillent sur des relations, deux nouvelles classes d'opérateurs sont nécessaires: les fenêtres, qui permettent de définir une relation à partir d'un flux (par exemple, les dix derniers tuples du flux), et les opérateurs de *streaming* pour l'opération inverse (par exemple, le flux des tuples insérés dans une relation).

D'autre part, c'est un système de gestion de services : il permet d'effectuer des appels de services à partir de requêtes orientées service. Les services sont vus comme des entités permettant l'invocation à distance de méthodes, de manière similaire aux *devices* UPnP. Les services sont représentés de manière homogène avec les données. Un service correspond en effet à un tuple dans une table étendue, dont certains attributs sont virtuels. Ces attributs virtuels représentent les entrées/sorties des méthodes des services de la table. Les valeurs de ces attributs virtuels ne sont pas enregistrées dans les tables, mais sont obtenues de manière dynamique en invoquant les services correspondants pendant l'exécution des requêtes continues à l'aide d'un opérateur spécifique à SoCQ, l'opérateur d'invocation.

Le moteur SoCQ permet ainsi d'exprimer des requêtes continues orientées service combinant les données classiques, les flux de données et des appels de services. Les requêtes SoCQ permettent d'exprimer de manière déclarative des applications pervasives utilisant ces sources de données non conventionnelles.

3.2 Gestion des ressources

La gestion des apparitions et disparitions de ressources est assurée par des gestionnaires de ressources. Ils sont également les intermédiaires entre SoCQ et les ressources. Cette organisation est illustrée en Figure 1.

Deux types de gestionnaires de ressources sont pris en compte : les gestionnaires de ressources en mode *push* et les gestionnaires de ressources en mode *pull*.

- Les gestionnaires de ressources en mode *push* fournissent au moteur SoCQ des données de façon continue. Ces gestionnaires sont typiquement utilisés pour envoyer au moteur SoCQ des flux de données provenant de capteurs. Le moteur SoCQ ne voit pas directement les ressources, mais seulement le flux de données résultat. Chaque flux de données peut provenir d'une source ou de plusieurs sources. Dans le second cas, le gestionnaire de données a pour rôle non

seulement de transmettre le flux au moteur SoCQ, mais également de construire un flux unique par une union des données fournies par l'ensemble des sources qu'il gère.

- Les gestionnaires de ressources en mode *pull* répondent à des sollicitations explicites du moteur SoCQ. Les ressources sont vues par le moteur SoCQ comme des services, permettant ainsi d'invoquer leurs méthodes à distance. Les gestionnaires permettent par exemple d'envoyer une commande à un dispositif physique de l'environnement (impression, réglage de chauffage, prise de photo…), de demander un calcul à un service distant, ou d'envoyer une requête à un SGBD. Chacune de ces sollicitations retourne au moteur SoCQ, par l'intermédiaire de son gestionnaire de ressources, l'ensemble de données résultant du traitement de l'action par le service.

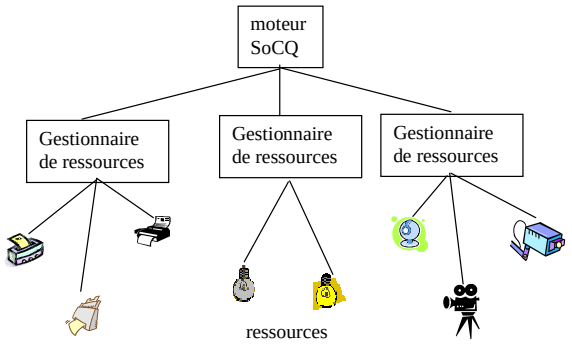

Figure 1 : Gestion de ressources

Les ressources sont réparties dans l'environnement et peuvent apparaître ou disparaître de façon imprévisible. Les gestionnaires de ressources doivent prendre en compte ces caractéristiques, et surtout les rendre transparentes au moteur SoCQ. Ainsi, un gestionnaire de ressources doit gérer et maintenir la liste des ressources disponibles, et agir à chaque instant en fonction de cette liste. Il doit pouvoir faire de la découverte dynamique des ressources, être informé lorsqu'une ressource apparaît ou disparaît du réseau.

Par exemple, lorsqu'un ensemble de capteurs de mouvement est présent dans l'environnement, le gestionnaire de ces ressources peut fournir au moteur SoCQ un flux de mouvements détectés. Ce flux contient l'union des mouvements détectés par les capteurs actifs. La liste de ces capteurs actifs peut fluctuer dans le temps, le moteur SoCQ reçoit dans tous les cas un flux unifié.

De la même façon, imaginons un gestionnaire de ressources qui maintient l'ensemble des imprimantes actives dans l'environnement. Lorsque ce gestionnaire reçoit une demande d'impression émanant du moteur SoCQ, il choisira un des services disponibles au moment de la demande.

Un gestionnaire de ressources peut gérer un ou plusieurs types de ressources. Cependant, chaque type de ressources sera géré de façon indépendante. Chaque ressource qui « arrive » dans l'environnement doit se déclarer. Le gestionnaire de ressources associé à ce type de ressources l'enregistre dans sa liste. Un mécanisme de gestion de la disparition d'une ressource est également nécessaire. Nous nous basons sur les mécanismes UPnP pour prendre en compte ces évolutions de l'environnement.

4. PROTOTYPE

Nous avons implémenté le moteur SoCQ en C++. Il est composé de trois modules principaux. Le module de gestion de requêtes est chargé de l'optimisation et de l'exécution des requêtes SoCQ, qui sont lancées par les utilisateurs à travers une interface en ligne de commande. Le module de gestion de tables permet d'une part au module de gestion de requêtes d'accéder aux données stockées, d'autre part aux sources de données externes de se connecter au système (par socket TCP) pour fournir leurs données. Le module de gestion de services est utilisé pour inscrire les services externes (ici les gestionnaires de ressources) auprès du système, afin que le module de gestion de requêtes puisse les appeler lors de l'exécution des requêtes SoCQ.

Les dialogues entre le moteur SoCQ et les gestionnaires de ressources sont implantés sous forme de sockets TCP. Les gestionnaires de ressources en mode *push* se comportent comme des clients TCP pour fournir les flux de données en continu. Les gestionnaires de ressources en mode *pull* se comportent comme des serveurs TCP qui attendent les commandes du moteur SoCQ. Le moteur SoCQ se comporte donc comme un serveur TCP dans le premier cas et comme un client TCP dans le second.

Dans cette architecture, nous avons également utilisé le protocole UPnP [22]. Les ressources sont des devices UPnP, les gestionnaires de ressources incluent des points de contrôle UPnP. Nous avons choisi ce protocole car il inclut une gestion des apparitions et disparitions de ressources, avec une description des services fournis par ces ressources.

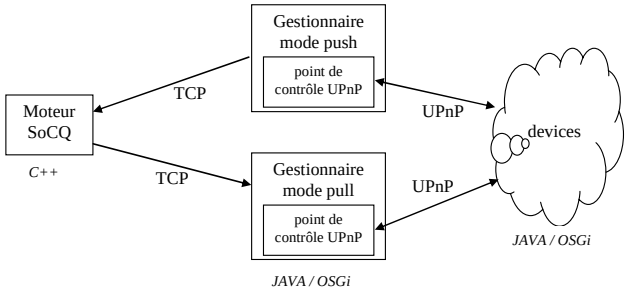

Figure 2 : Architecture du prototype

Les gestionnaires de ressources et les ressources ont été développés dans le framework OSGi [21] avec son implantation Felix [9]. OSGi est un outil de gestion d'applications évolutives dont l'unité de déploiement et de gestion est le bundle. Un bundle est un ensemble de classes Java. Chaque device et chaque gestionnaire de ressources fait l'objet d'un bundle. Ainsi, l'apparition et la disparition de devices est non seulement facilement reconnue par les gestionnaires de ressources grâce à UPnP, mais est également gérée de façon fine par les outils de gestion du cycle de vie des bundles offert par OSGi. L'ajout ou le retrait de gestionnaires de ressources est également très facile à faire, même à chaud. OSGi a également l'intérêt de fournir une bibliothèque de développement UPnP qui rend transparentes toutes les problématiques de développement UPnP.

La Figure 2 présente une vue globale de l'architecture de notre prototype.

5. SCENARIO ET EXPERIMENTATIONS

Nous avons construit et mis en œuvre un scénario permettant la validation des principes définis. Notre scénario concerne la surveillance de températures dans plusieurs lieux. Dès lors qu'une température dépasse un certain seuil, une photo du lieu concerné est prise et un message est envoyé à un utilisateur (responsable de la zone surveillée). Ce scénario implique les ressources suivantes : des capteurs de température, des caméras (actionneurs de prise de photos), un service d'envoi de messages instantanés.

Les capteurs de température utilisés sont des Maxim Dallas de type Thermochron® iButton DS1921 [20] (version USB). Les caméras sont de deux types : IP Axis 241 Network Camera (caméra IP avec serveur web intégré) ou webcam Logitech (USB). Un device UPnP logiciel offre un service de messagerie basé sur Jabber [24].

Un gestionnaire de températures récupère périodiquement les températures relevées par les capteurs. Il est en mode *push* : ce gestionnaire de ressources construit un flux continu qu'il envoie au moteur SoCQ. Un gestionnaire de caméras permet de lancer une prise de photo sur une caméra donnée ; il est en mode *pull*. Un second gestionnaire de ressources mode *pull* permet d'appeler le service d'envoi de messages. Le gestionnaire de caméras comporte également une servlet permettant de stocker les photos prises et de les retourner sur demande (clic par l'utilisateur sur l'url dans le corps du message reçu).

La traduction de ce scénario dans le framework SoCQ se compose de deux éléments : la description des ressources (catalogue) et la description de l'application (requête).

La description des ressources est la suivante :

```
STREAM Temperature (
        Area CHAR,
        Temperature REAL
)
RELATION Surveillance (
        Area CHAR,
        Manager STRING(20),
        Threshold REAL,
        AlertMessage STRING(20)
)
RELATION Employee (
        Name STRING(20),
        Address STRING(50),
        Messenger SERVICE,
        Message STRING(20) VIRTUAL,
        URL String(100) VIRTUAL,
        Sent CHAR VIRTUAL
)
RELATION Camera (
        Camera SERVICE,
        Area CHAR VIRTUAL,
        UrlPhoto STRING(100) VIRTUAL
)
```

Temperature représente le flux de températures. Il contient les localisations des capteurs (*area*) et les valeurs des températures (*temperature*). *Surveillance* est une table relationnelle classique dont chaque occurrence décrit une localisation à surveiller (*area*), le nom du responsable de la localisation (*manager*), le seuil de température (*threshold*) et le début du texte du message d'alerte à envoyer (*alertMessage*). *Employee* est une table relationnelle contenant des attributs classiques et des attributs virtuels (qui seront valués à l'appel de services au cours du traitement de la requête). Elle contient les noms des employés (*name*), leurs adresses de messagerie (*address*), le service de messagerie à appeler (*messenger*), deux attributs virtuels d'entrée correspondent au corps du message (*message*) et à l'URL de photo à envoyer dans le message (*URL*) et un attribut virtuel de sortie correspondant à la confirmation de l'envoi du message (*sent*). *Camera* est une table relationnelle permettant de faire référence à un service de prise de photo. Elle a un attribut d'entrée virtuel (*area*) permettant de donner la localisation à prendre en photo, et un attribut de sortie virtuel fournissant l'url de la photo prise (*URLPhoto*).

La requête rédigée dans le framework SoCQ est la suivante :

```
SELECT Area, Manager, Threshold, Temperature, URL, Sent
FROM Temperature T, Surveillance S, Employee E, Camera C
WHERE S.Manager = E.Name
AND S.Area=T.Area
AND C.Area=T.Area
AND S.Threshold<T.Temperature
AND E.Message=S.AlertMessage
AND E.URL=C.URLPhoto
```

Ce requête SoCQ compare les températures avec les seuils enregistrés dans *Surveillance*. Si les valeurs sont supérieures aux seuils, le moteur SoCQ fait appel au gestionnaire de caméras pour qu'il invoque le service de prise de photo adéquat. A réception de l'url de la photo, le moteur SoCQ peut faire appel au gestionnaire de messagerie qui invoque le service requis et retourne son acquittement (attribut *sent*).

La copie d'écran de la Figure 3 montre l'ensemble des messages reçus par un utilisateur suite à des dépassements de seuils de températures.

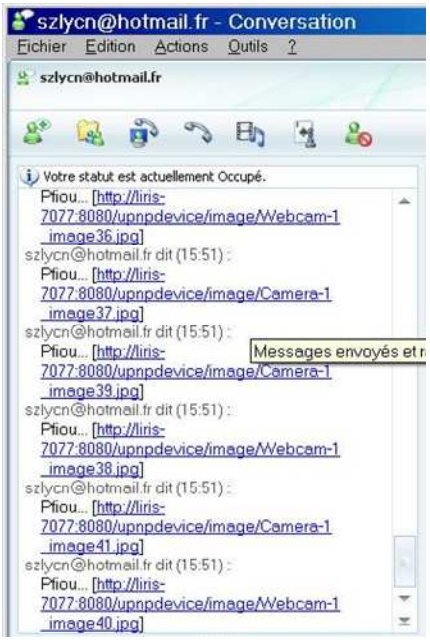

Figure 3 : Messages instantanés reçus suite à des dépassements de seuils de températures (client de messagerie MSN Messenger)

Nous avons réalisé quelques tests de performances pour les gestionnaires de ressources mode *pull*. Nous avons voulu évaluer

l'impact de la répartition des différents modules de notre architecture sur le temps que le moteur SoCQ passe à attendre les réponses des invocations de ressources. Nous avons relevé des valeurs pour deux services : getImage (utilisé par le gestionnaire de caméras) et sendMessage (utilisé par le gestionnaire de messagerie). Ceci ne concerne pas le point de contrôle mode *push* : sa période d'envoi des flux est fixée par l'utilisateur. Nous avons considéré quatre configurations selon la répartition des composants que sont le moteur SoCQ, gestionnaires de ressources et devices :

- centralisée : tous les composants sont sur la même machine,
- répartie : chaque composant est sur une machine distincte,
- devices distants : le moteur SoCQ et les gestionnaires de ressources sur la même machine, les devices sont sur des machines distantes,
- SoCQ distant : les devices et les gestionnaires de ressources sur la même machine, le moteur SoCQ sur une machine distante.

Les machines sont des PC standards, dans un même LAN avec liaisons TCP/IP. Pour mesurer le temps d'appel, chaque action est invoquée 10 fois de suite par le moteur SoCQ. Cette opération est effectuée 3 fois afin d'obtenir la moyenne pour chaque action et chaque configuration. Pour sendMessage, le message envoyé est « Hello, il fait trop chaud ». Pour getImage, les tests sont effectués sur une webcam. Les résultats de ces tests sont donnés en Figure 4.

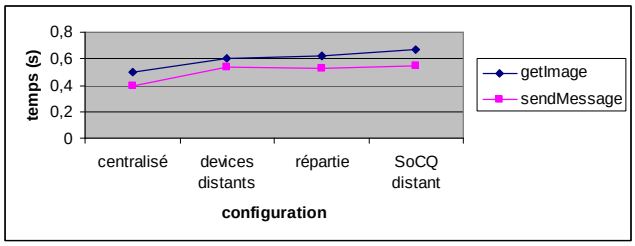

Figure 4 : Temps des actions

Dans la configuration centralisée, le temps d'une invocation est plus petit que dans les trois autres cas, car la transmission réseau nécessite du temps (0.2 à 0.3 s). Pour sendMessage, le temps d'invocation est d'environ 0.5s. Cependant, le client de messagerie reçoit les messages 2 ou 3s plus tard car bien que l'action soit effectuée et terminée, les messages ne sont pas encore arrivés jusqu'au client à cause de la transmission réseau et du protocole de communication de messagerie XMPP.

La configuration centralisée et la configuration « SoCQ distant » ne sont pas réalistes, car les devices UPnP sont toujours répartis. Le facteur sur lequel on peut agir est la position des gestionnaires de ressources, qui peuvent être sur une même machine que le moteur SoCQ ou non. Selon nos essais, il n'y a pas de différence significative au niveau des performances, donc nous pouvons choisir la structure en fonction des contraintes matérielles.

Nous avons aussi testé le temps de découverte dynamique des devices. Lorsqu'un device distant du gestionnaire de ressources rejoint le réseau, il faut environ 45 à 50 secondes pour que le point de contrôle le détecte et l'ajoute dans sa table. Nous ne pouvons pas améliorer ce temps très long, car il dépend du protocole UPnP et de l'implantation du bundle UPnP BaseDriver de Felix, qui mériterait certainement des optimisations.

6. CONCLUSION ET PERSPECTIVES

Les environnements pervasifs sont caractérisés entre autres par la dynamicité et l'hétérogénéité des ressources environnantes. Dans le projet SoCQ, nous avons pris le parti de fournir à l'utilisateur concepteur d'applications une vue uniforme et orientée données de cet environnement. La construction d'une application se fait donc par la rédaction de requêtes dans un langage déclaratif. Dans notre architecture, la dynamicité est prise en compte par les gestionnaires de ressources et leur implantation en UPnP.

Pour l'instant, nos gestionnaires de ressources sont typés, c'est-à-dire qu'ils ne prennent en charge qu'un seul type de ressource, type qu'ils connaissent à l'avance. Nous avons commencé à concevoir et mettre en place des gestionnaires de ressources génériques, qui sachent prendre en compte toute ressource, que son type soit déjà connu ou non.

Après cette étape, il sera intéressant et utile d'étudier une collaboration plus étroite entre les gestionnaires de ressources et le moteur SoCQ, afin que ce dernier puisse prendre en compte dans son catalogue les nouveaux types de ressources découverts. Le framework SoCQ fournira alors une représentation de l'environnement tel qu'il est, et non tel qu'il est attendu.

7. REFERENCES

[1] D. J. Abadi et al. The Design of the Borealis Stream Processing Engine. In CIDR 2005, Proceedings of Second Biennial Conference on Innovative Data Systems Research, 2005.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford Stream Data Manager. IEEE Data Engineering Bulletin, 26(1):19–26, 2003.

[3] C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM – A Component System for Pervasive Computing. In PerCom'04, Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications, page 67, 2004.

[4] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. EasyLiving: Technologies for intelligent environments. In HUC 2000, Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing, pages 12–29, 2000.

[5] S. Chandrasekaran et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research, 2003.

[6] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 379–390, 2000.

[7] M. Cherniack et al. Scalable Distributed Stream Processing. In CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research, 2003.

[8] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. IEEE Pervasive Computing, 1(1):59–69, 2002.

[9] Felix. Apache Felix Project. http://felix.apache.org/.

[10] M. Franklin, A. Halevy, and D. Maier. From Databases to Dataspaces: a new Abstraction for Information Management. SIGMOD Rec., 34(4):27–33, 2005.

[11] M. Franklin et al. Design Considerations for High Fan-In Systems: The HiFi Approach. In CIDR 2005, Proceedings of Second Biennial Conference on Innovative Data Systems Research, 2005.

[12] D. Garlan et al. Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing, 1(2):24–31, 2002.

[13] R. Grimm et al. System Support for Pervasive Applications. ACM Transactions on Computer Systems, 24(4):421–486, November 2004.

[14] Y. Gripay, F. Laforest, and J.-M. Petit. Vers les Requêtes Continues orientées Actions dans les Systèmes Pervasifs. In BDA'07, Proceedings of Bases de Données Avancées, 2007.

[15] Y. Gripay. Service-oriented continuous queries for pervasive systems. In EDBT'08. Proc. of the 11th int. Conf. on Extendinhg DataBase Technology, PhD Workshop, 2008

[16] Y. Gripay et al. The SoCQ project web site. http://socq.liris.cnrs.fr

[17] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura,E. Jansen, The Gator Tech Smart House: a programmable pervasive space, *Computer* , 38(3):50–60 , 2005.

[18] T. Imielinski and B. Nath. Wireless graffiti: data, data everywhere. In VLDB'2002: Proceedings of the 28th international conference on Very Large Data Bases, pages 9–19. VLDB Endowment, 2002.

[19] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative support for sensor data cleaning. In Pervasive, pages 83–100, 2006.

[20] Maxim. http://www.maxim-ic.com/products/ibutton/products/ibuttons.cfm#sensor

[21] OSGi. OSGi Alliance. http://www.osgi.org/.

[22] UPnP. Universal Plug and Play. http://www.upnp.org/.

[23] M. Weiser. The Computer for the 21st Century. Scientific American, 265(3):94–104, September 1991.

[24] XMPP. Extensible Messaging and Presence Protocol. http://www.xmpp.org/.

[25] W. Xue and Q. Luo. Action-Oriented Query Processing for Pervasive Computing. In CIDR 2005, Proceedings of the Second Biennial Conference on Innovative Data Systems Research, 2005.

[26] Y. Yao and J. Gehrke. Query Processing in Sensor Networks. In CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research, 2003.

[27] F. Zhu, M. Mutka, and L. Ni. Service Discovery in Pervasive Computing Environments. IEEE Pervasive Computing, 4(4):81–90, 2005.