# Gestion dynamique de contexte pour l'informatique diffuse*

# Dynamic context management for pervasive computing

Jérôme Euzenat[1] Jérôme Pierson[2] Fano Ramparany[2]

[1]INRIA Rhône-Alpes
[2]France Telecom R&D

Jerome.Euzenat@inrialpes.fr, {Jerome.Pierson,Fano.Ramparany}@francetelecom.com

## Résumé

*L'informatique diffuse a pour but d'offrir des services aux utilisateurs humains interagissant avec leur environnement (y compris les objets et autres humains qui l'occupent). Les applications dans ce domaine doivent être capable de considérer le contexte dans lequel les utilisateurs évoluent (qu'il s'agisse de leur localisation physique, leur position sociale ou hiérarchique ou leurs tâches courantes ainsi que des informations qui y sont liées). Ces applications doivent gérer dynamiquement l'irruption dans la scène de nouveaux éléments (utilisateurs ou appareils) même inconnus et produire de l'information de contexte utile à des applications non envisagées.*

*Après avoir examiné les différents modèles de contexte étudiés en intelligence artificielle et en informatique diffuse, nous montrons en quoi ils ne répondent pas directement à ces besoins dynamiques. Nous décrivons une architecture dans laquelle les informations de contexte sont distribuées dans l'environnement et où les gestionnaires de contexte utilisent les technologies développées pour le web sémantique afin d'identifier et de caractériser les ressources disponibles.*

*L'information de contexte est exprimée en RDF et décrite par des ontologies en OWL. Les dispositifs de l'environnement maintiennent leur propre contexte et peuvent communiquer cette information à d'autres dispositifs. Ils obéissent à un protocole simple permettant de les identifier et de déterminer quelles informations ils sont susceptibles d'apporter. Nous montrons en quoi une telle architecture permet d'ajouter de nouveaux dispositifs et de nouvelles applications sans interrompre ce qui fonctionne. En particulier, l'ouverture des langages de description d'ontologies permettent d'étendre les descriptions et l'alignement des ontologies permet de considérer des ontologies indépendantes.*

## Mots Clef

Informatique diffuse, Contexte, Web sémantique, OWL, Hypothèse du monde ouvert, Composants.

## Abstract

*Pervasive computing aims at delivering services based on the opportunity for human beings to interact with their environment (including the objects and humans populating it). Applications must be able to take into account the context in which users evolve (e.g., physical location, social or hierarchical position, current tasks as well as related information). These applications have to deal with the dynamic integration of new, and sometimes unexpected, elements (users or devices), in the environment. Furthermore, the environment has to provide context information to newly designed applications.*

*After considering context models developed in pervasive computing and artificial intelligence, we conclude that they fall short in addressing these dynamic needs. We describe an architecture in which context information is distributed in the environment and where context managers use semantic web technologies in order to identify and characterize available resources.*

*Context information is expressed in RDF and described through OWL ontologies. The components in the environment maintain their own context and can communicate this information to other components. They obey a simple protocol for identifying them and for determining the information they are able to provide. We show how this architecture allows the introduction of new components and new applications without disrupting ongoing processes. In particular, the openness of ontology description languages enables the extension of context descriptions and ontology matching helps dealing with independently developed ontologies.*

## Keywords

Pervasive computing, Context, Semantic web, OWL, Open-world assumption, Components.

* Les travaux de Fano Ramparany et Jérôme Pierson sont financés en partie par le projet Européen Amigo (IST-2004-004182) ; le travail de Jérôme Euzenat est financé en partie par le projet Européen Knowledge Web (IST-2004-507482).

# 1. Introduction

Dans un environnement d'informatique diffuse ("pervasive computing") divers services élémentaires peuvent être offerts par des dispositifs physiques (capteurs, actionneurs, dispositifs matériels d'interface humaine pour l'acquisition et la restitution d'information). Des services plus élaborés sont fournis par des dispositifs actifs pouvant aller de la simple agrégation de services (comme le calcul d'une température moyenne entre toutes celles renvoyées par des capteurs) à l'application d'assistance aux utilisateurs finaux impliquant la communication avec de nombreux services.
La problématique de l'informatique diffuse implique la collaboration de dispositifs hétérogènes. Ils doivent agir en fonction de leur environnement et celui de leurs utilisateurs. Il existe des éléments d'information que l'on peut considérer comme transversaux à tous les services. C'est particulièrement le cas de la localisation spatiale, mais aussi d'autres éléments relatifs à l'état de l'environnement physique dans lequel les services sont situés [14, 18]. Ces éléments font partie du contexte dans lequel les applications sont invoquées.

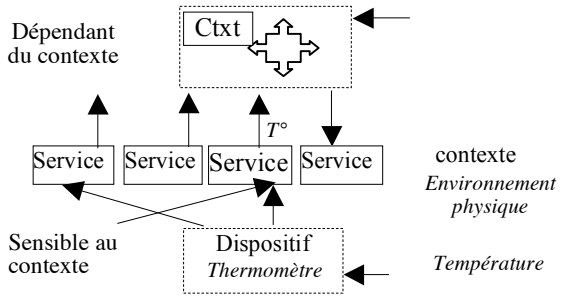


Figure 1: Architecture générale d'un environnement d'informatique diffuse: des dispositifs offrent des services de plus en plus élaborés permettant d'appréhender le contexte d'une tâche. Ce dernier doit être géré par certains dispositifs. Les mentions en *italiques* constituent un exemple.

La figure 1 synthétise les influences que les dispositifs, les services et le contexte ont les uns sur les autres. Nous utiliserons dans la suite de vocabulaire suivant : L'*environnement* dénotera tout, c'est-à-dire l'environnement physique dans lequel se trouve l'utilisateur, l'ensemble des dispositifs qui y sont présents ainsi que la description complète de la situation. Par *dispositif*, on entendra tout dispositif matériel présent dans l'environnement capable de communiquer. Un *capteur* sera un dispositif sensible au contexte (context-sensitive) capable de caractériser un aspect de l'environnement physique alors que l'*application* sera un programme capable de fournir un service à un utilisateur. On s'intéressera ici uniquement à des applications dépendantes du contexte (context-aware). Les applications que nous considérons plus spécialement sont tout d'abord intéressées à l'appréhension du contexte physique (on suppose que ces applications ont identifié les tâches en cours d'accomplissement), cependant, on verra qu'elles ont toujours la possibilité de recueillir des informations dans un cadre plus global et, en particulier, elles exploiteront le web.

Bien que non central à la fonction d'un service, le contexte conditionne en général son choix ou son fonctionnement. Ainsi, on ne peut coordonner une caméra et un microphone que s'ils sont dans une même pièce et l'accessibilité d'un utilisateur à un dispositif physique est souvent primordiale dans la sélection de celui qui fournira le service requis. D'autre part, en fonction du contexte, les services invoqués doivent être adaptés (pour acheminer physiquement le résultat auprès du bénéficiaire ou tout simplement pour rendre le résultat dans une langue et un format appropriés).
Tenir compte de ce contexte pour produire des solutions adaptées à l'environnement d'un utilisateur est l'un des buts principaux de l'informatique diffuse. Jusqu'à présent les solutions proposées sont relativement statiques et nécessitent pour chaque nouvelle application la redéfinition de ce qui en constitue le contexte [9, 10, 13, 16, 21]. Bien que celui-ci soit effectivement dépendant de la tâche à accomplir et donc de l'application, on retrouve un certain nombre d'éléments, tels que la localisation dans l'espace, évoqué plus haut, commun à un certain nombre d'entre-elles.

Notre but est donc de concevoir un système de gestion de contexte (a) suffisamment général pour être utilisé par différentes applications d'informatique diffuse, (b) suffisamment spécifique pour couvrir les informations fournies par les dispositifs déjà utilisés comme les services de localisation, (c) suffisamment flexible pour accepter et exploiter dynamiquement l'introduction de nouveaux dispositifs.

On se propose donc d'abstraire cette notion de contexte afin de spécifier l'adaptation au contexte de manière indépendante d'une application particulière. On proposera pour cela une représentation des informations de contexte indépendante des applications et l'architecture permettant de les acquérir (au préalable, puis dynamiquement). L'ouverture du système impliquera l'introduction de descriptions hétérogènes qui devront être réconciliées avant de pouvoir être exploitées. Pour cela on s'appuiera sur les techniques développées dans le cadre du « web sémantique ».

Nous examinons d'abord les travaux très divers couvrant le champ de la gestion de contexte en informatique diffuse et en intelligence artificielle (§2). Nous expliquons en quoi ils ne répondent pas complètement aux besoins évoqués plus haut. Nous proposons alors une solution à ce problème fondée sur :

— Une représentation générale des informations de contexte fondée sur les langages du web sémantique (§3) ;
— Une architecture permettant de distribuer les informations de contexte de telle sorte qu'il soit aisé d'introduire de nouveaux dispositifs produisant et/ou utilisant cette information (§4) ;
— Un formalisme permettant d'accéder à cette information et de l'exploiter tout en permettant tou-

jours d'étendre la représentation disponible d'un contexte (§5) ;

— Un dispositif permettant de tirer parti de l'information disponible, même si elle n'a pas été envisagée lors de la mise en place de l'application (§6).

Tout au long de cet article nous utiliserons un scénario très simple. Décrit ici informellement, il sera reconsidéré à chaque étape pour montrer comment il est effectivement possible de l'implémenter. Une personne est tranquillement installée dans sa chambre d'hôtel lorsqu'elle reçoit un appel téléphonique. L'appareil téléphonique, constatant que l'appel n'est nullement urgent va chercher à savoir s'il doit émettre un signal sonore, une vibration ou dérouter l'appelant vers une messagerie. Afin de préserver au mieux la tranquillité de son utilisateur, il cherchera à connaître son état (sommeil, discussion avec un tiers, etc.). Pour cela, il va chercher à tirer parti des informations que l'environnement est capable de lui fournir: autres appareils électriques en fonctionnement (ordinateur, sèche cheveux, téléviseur), présence de tiers dans la pièce, état physique de l'environnement (température, degré hygrométrique, luminosité). On peut se rendre compte, à partir des exemples donnés, que le support à l'application téléphonique peut se trouver dans de nombreuses chambres d'hôtel. Mais elles n'ont pas été prévue (a) pour communiquer entre elles, (b) pour le type d'application envisagé .

On pourrait résumer notre objectif comme proposer une implémentation de (a) faisant en sorte que (b) ne soit jamais nécessaire.

## 2. Les contextes

Le contexte est l'ensemble des informations caractérisant (partiellement) la situation d'une entité particulière [9]. C'est une notion dont l'acception n'est ni absolue ni universelle, mais relative à une situation [11, 3]. Cette situation peut être une situation physique (comme la localisation spatio-temporelle d'une personne) ou fonctionnelle comme la tâche en cours de réalisation.

Comme le contexte n'est pas contraint à contenir toute l'information caractérisant une situation (ce n'est sans doute pas possible), il y a place pour plusieurs contextes pour une même situation. D'autre part, comme ces contextes peuvent être comparables en fonction des ensembles d'information qu'ils contiennent, un contexte peut être considéré comme plus ou moins étendu qu'un autre.

D'autre part, cette individualisation du contexte en un ensemble d'éléments de contexte est nécessaire. En effet, une application d'informatique diffuse doit être capable d'entrer et de sortir des contextes : lorsque l'utilisateur entre dans un bâtiment, le contexte associé à la ville dans laquelle il se situe doit passer en arrière plan alors que celui concernant l'intérieur du bâtiment doit devenir principal (pour être oublié lors de la sortie du bâtiment).

Ainsi les informations de température, de pression, d'ensoleillement (luminosité), d'humidité et de temps (date et heure) sont des éléments de contexte. Si on souhaite implémenter un service de thermomètre, le contexte se limitera à la température. Si on souhaite implémenter un thermostat ou un service de climatisation, on agrégera les informations de température, d'ensoleillement et de temps pour réguler la température. Enfin, le contexte d'un service de station météo regroupera les cinq informations. Chaque ensemble de la figure 2 est un contexte et peut être nécessaire à différentes applications. Il y a aussi d'autres contextes possibles combinant ces informations brutes.

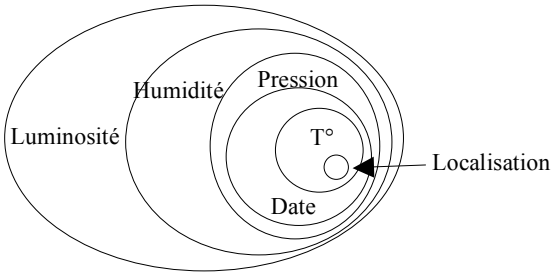


Figure 2: Le contexte applicatif d'une centrale de climatisation.

Bien que les travaux dans divers domaines se soient penchés sur la notion de contexte, les points de vue sous lesquels cette notion est abordée sont différents. En informatique diffuse, on s'est beaucoup intéressé au contexte d'une application en termes des paramètres physiques qui caractérisent sa situation. En communication homme-machine, le contexte est plus souvent la tâche que l'utilisateur désire accomplir et l'histoire de l'interaction avec le système dans ce but [11]. En intelligence artificielle, le contexte est plutôt considéré comme les conditions qui font qu'une assertion est valide où ne l'est pas [15].

### 2.1. Le contexte en informatique diffuse

En informatique diffuse, le contexte physique est une donnée de première importance ; il est acquis à partir de données de capteurs en général. Ces données sont élaborées en des caractérisations du contexte plus adaptées à leur utilisation (par exemple, « température élevée » dans le cadre du contrôle de la climatisation). Par rapport à la donnée brute produite par le capteur, celle-ci est appauvrie (c'est-à-dire moins précise) mais plus adaptée.

Les différentes définitions du contexte en informatique diffuse sont ainsi très souvent liées à une application ou, au mieux, à un domaine particulier [10, 4]. Cette caractérisation du contexte a pour défaut sa dépendance à la tâche que l'on veut accomplir. « Température élevée » n'est pas un terme absolu, il dépend de l'utilisation que l'on veut faire de la pièce, comme sauna ou comme chambre à coucher. Plus que de contexte, en informatique diffuse, on manipule une caractérisation du contexte vis-à-vis d'une application. Il sera donc peu aisé d'implémenter dynamiquement de nouvelles applications à l'aide de la caractérisation d'un contexte qui a été développé en fonction d'une application particulière.

Cependant, la préoccupation de modélisation multi-applications du contexte est maintenant reconnue. [6] pose le

problème de considérer cette notion de contexte comme n'étant pas prédéfinie par les applications. Ainsi, le diagramme de la figure 3 présente une manière d'accommoder progressivement les informations de contexte : des capteurs à l'application. Nous suivrons aussi cette voie en précisant le contenu des couches perception et situation de telle sorte qu'elles puissent supporter la mise à jour dynamique de l'environnement.

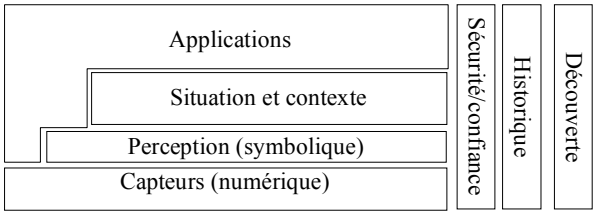


Figure 3: Modèle de contexte en informatique diffuse. Les données provenant des capteurs sont agrégées et élaborées pour obtenir le modèle de contexte utilisé par les applications (d'après [6]). Ici, nous ne considérons pas les aspects transversaux (découverte, historique et sécurité).

## 2.2. Le contexte en intelligence artificielle

En intelligence artificielle, la notion de contexte concerne, en général, la représentation de l'information. La notion de contexte est exploitée pour rendre compte de deux aspects conjugués : le contexte d'émission de l'information (et plus précisément son contexte de validité [8]) et le fait que plus on développe un raisonnement dans un contexte spécifique, plus on sera efficace [14].

John McCarthy [20] a proposé une formalisation logique du contexte fondé sur une « réification » du contexte ainsi que sur un « méta-prédicat » *ist*, *ist*(*p*,*c*) signifiant que l'assertion *p* est vraie dans le contexte *c*. Les approches du contexte en intelligence artificielle permettent donc de grouper la connaissance en micro-théories [14] et de se placer dans ou hors du contexte de ces théories afin de raisonner. Dans le cadre de Cyc, le contexte permet de fournir un cadre plus précis pour l'interprétation d'une information.

Ce type de fonctionnement peut être utilisé en informatique diffuse afin d'intégrer et d'interpréter les données issues des capteurs. S'adjoindre la théorie associée au capteur permet de tirer parti de l'information qu'il produit sans ambiguïté. Dans cette optique, les données brutes, issues de capteurs ne sont généralement pas appauvries, mais plus souvent agrégées (et associées à d'autres informations permettant de les interpréter plus précisément).
[15] propose un moyen d'exprimer ce type de contexte pour le web sémantique en associant à chaque triplet une information sur sa provenance ("quad"). il s'agit ici d'être capable d'interpréter l'information brute provenant de sources diverses (de type fil de nouvelle). D'autres extensions sur le même modèle ont été proposées [18] et sont implémentées dans les gestionnaires de RDF modernes.

Alors que les travaux de McCarthy et Guha consistent à considérer les contextes comme des théories indépendantes portant chacune sur un champ particulier de la connaissance, Fausto Giunchiglia considère plutôt les contextes comme des points de vue concurrents sur une même information. Il explicite les relations entre contextes sous la forme de transformations ("mappings") qui permettent d'importer dans un contexte l'information qui se trouve dans un autre. Cette approche peut aussi être utile en informatique diffuse lorsque plusieurs sources d'information produisent des informations comparables. Ces travaux ont aussi, sous la forme du langage C-OWL une incarnation liée aux langages du web sémantique [2]. Une comparaison des deux approches est donnée dans [24].

## 2.3. Synthèse

Si l'on résume les deux approches précédentes on peut considérer que l'informatique diffuse a une approche situationnelle du contexte alors que l'intelligence artificielle a une approche informationnelle. Plus notable, en informatique diffuse, le contexte est très souvent formé autour des besoins d'une application particulière alors qu'en intelligence artificielle, c'est plutôt la source de l'information qui en détermine le contexte. Enfin, en informatique diffuse, l'information provenant des sources a tendance à être plutôt appauvrie pour être directement adaptée à l'application alors que l'intelligence artificielle tend plutôt à agréger l'information de contexte sans l'appauvrir.

Bien entendu, rien n'est si tranché et les deux approches sont plutôt complémentaires qu'opposées. Les données brutes dans les deux cas peuvent subir appauvrissement et agrégation. On peut le voir dans la figure 4.

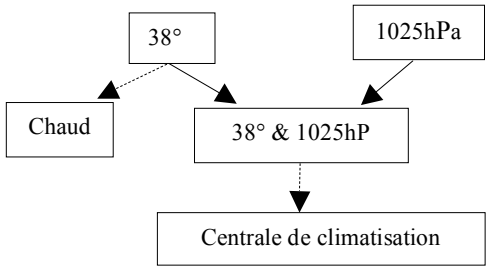


Figure 4: Contexte, agrégation (traits pleins) et appauvrissement (traits pointillés). Les informations fournies par les capteurs devraient pouvoir être manipulées et transformées suivant le besoin des applications.

Le pont entre les deux approches est effectivement d'actualité. Ainsi, dans [6], la question de l'évolution des modèles de contexte est posée. C'est à cette question que nous cherchons à répondre ici.

Notre but est de développer une architecture permettant la prise en compte dynamique de nouveaux éléments de contexte (dispositifs engendrant de l'information de contexte) et d'introduire de nouvelles applications tirant parti de l'information disponible sans interruption. Pour

cela on propose une architecture de gestion de contexte fondée sur des composants, un formalisme pour représenter le contexte fondé sur les technologies développées pour le web sémantique et on montre comment ils peuvent être utilisés pour étendre dynamiquement l'environnement.

## 3. Modéliser le contexte dans le web sémantique

Nous introduisons ici le modèle simple que nous allons utiliser pour représenter les informations de contexte nécessaires aux applications d'informatique diffuse (§3.2). Ils sont basés sur les langages du web sémantique que nous introduisons d'abord (§3.1). Leur utilisation dans le contexte des applications qui nous occupent sera justifié par la suite (§5 et 6).

### 3.1. RDF et OWL

Le langage de base du web sémantique est RDF (Resource Description Framework [19]). Il permet d'exprimer des assertions de type sujet-prédicat-objet (appelés triplets et notés <*s p o*>). C'est donc un sous-ensemble du calcul des prédicats. On peut aussi voir un tel ensemble de triplets comme un multi-graphe étiqueté et ainsi faire le lien avec le formalisme des graphes conceptuels. La force de RDF est que les noms des entités (qu'ils soient sujets, prédicats ou objets) sont des URI (les identificateurs du web, que l'on peut voir comme une généralisation des URL : http://www.w3.org/sw). Il est ainsi possible dans plusieurs documents RDF de faire référence à une même entité avec certitude (on peut raisonnablement supposer qu'un URI dénote la même chose quelque soit son utilisateur).
Le langage OWL [7], pour sa part, permet de décrire des « ontologies » ou modèle conceptuel d'un domaine particulier et ainsi de permettre plus facilement l'interprétation des graphes RDF concernant ce domaine. Il permet de définir des classes d'objets et de prédicats ainsi que de déclarer des contraintes pesant sur ces objets.
Il n'existe pas à proprement parler d'outil de gestion du contexte au sens ou nous l'entendons dans le web sémantique (il existe des propositions comme celles de CDF [18], [15] ou C-OWL [2]). Mais il se trouve que les langages développés pour le web sémantique, et en particulier RDF et OWL, sont adaptés à la représentation du contexte en informatique diffuse et plus particulièrement à une représentation du contexte capable de supporter les aspects dynamiques. Ceci pour deux raisons :

— Ces langages sont ouverts : ils implémentent l'hypothèse du monde ouvert où il est toujours possible d'avoir plus d'information que l'on en dispose à un moment donné. Ceci est adapté à un environnement dynamique où une information momentanément indisponible est de nouveau accessible. C'est aussi très adapté, on le verra plus loin, à un monde où l'on ajoute dynamiquement de nouvelles représentations.
— Ces langages sont destinés à fonctionner en réseau. Pour cela ils disposent d'un espace d'adressage global (les URI) qui permettent de fusionner des représentations sans risque de conflits de noms.

Nous utiliserons donc ces langages dans les représentations de contextes que nous définirons par la suite.

### 3.2. Un modèle de contexte

Le modèle de contexte utilisé à ce stade est très simple : un contexte sera un ensemble d'assertions RDF. Les éléments de contexte seront donc considérés comme du RDF.
L'avantage d'utiliser un langage de la généralité de RDF est que l'on peut exploiter une interface de très haut niveau et que les informations de contexte stockées et communiquées peuvent être simplement considérées comme des triplets. L'interopérabilité est alors simplement garantie en considérant que les composants gérant le contexte sont des consommateurs et des producteurs de RDF.
Ceci n'est cependant pas suffisamment précis et il est souhaitable de n'extraire que l'information utile des différentes sources d'information de contexte disponibles. Pour cela on pourra utiliser un langage comme RDQL [23] ou SPARQL permettant de poser (ou de s'abonner à) des requêtes précises aux composants.
Pour pouvoir poser les bonnes requêtes aux bons composants, il est nécessaire pour les composants de rendre public les requêtes auxquelles ils sont capables de répondre. Cela peut être réalisé par la publication des classes d'objets et les propriétés sur lesquelles le composant peut répondre. Le langage OWL est tout à fait adapté pour spécifier (par le biais d'une ontologie) ces éléments.
On ne considèrera pas à ce stade de réification des contextes ni d'opérations de manipulation de contextes (mais ceci devra pouvoir s'insérer ultérieurement dans l'architecture proposée). En l'état, ces manipulations seront prises en charge par l'architecture qui permettra aux dispositifs d'acquérir leur contexte en combinant ceux des autres dispositifs.

### 3.3. Exemple

Une partie des informations concernant la chambre d'hôtel peut être représentée par l'ensemble des triplets suivants :

| | | |
|---|---|---|
| < self | ami:location | #1345 > |
| < #1345 | rdf:type | ami:HotelRoom > |
| < #67 | ami:location | #1345 > |
| < #67 | rdf:type | tv:tvset > |
| < #67 | tv:state | tv:on > |
| < #67 | tv:volume | tv:muted > |
| < #67 | tv:channel | tv:arte > |

Ils signifient que l'application est localisée (sémantiquement) dans une chambre d'hôtel contenant par ailleurs un poste de télévision diffusant Arte avec le volume coupé. On remarque le typage des entités (tvset, HotelRoom) ainsi que l'assemblage de vocabulaires utilisant des espaces de nom différents (rdf, ami, tv).

## 4. Service de gestion du contexte

Afin de disposer des informations de contexte, les applications d'informatique diffuse doivent les puiser directement ou indirectement des capteurs en relation avec l'environnement. Nous proposons une architecture permettant de réaliser ceci de la manière la plus flexible possible, c'est-à-dire en évitant que chaque application ait à contacter directement chaque capteur et que l'ajout d'un capteur nécessite la recompilation de toutes les applications.

### 4.1. Architecture

Envisager une architecture où les services sont attentifs au contexte (context-aware), conduit tout naturellement à proposer un service de gestion du contexte capable de servir aux autres dispositifs les informations de contexte.
On l'a vu (Figure 1), le recueil d'information sur le contexte physique se fait par le biais de capteurs (de température, de luminosité, etc.). Ces capteurs n'ont en général pas la capacité à gérer explicitement un contexte, on considèrera qu'ils sont vus par les systèmes d'informatique diffuse au travers de proxys capables de fournir les informations saisies par les capteurs. La première possibilité consiste à considérer que les applications communiquent directement avec les capteurs dont elles ont besoin. Cette architecture directe a l'inconvénient d'obliger les applications à savoir à qui s'adresser pour obtenir l'information nécessaire. Elle complexifie l'agrégation des informations (qui doit être réalisée à chaque fois par l'application) et ajouter à la charge des dispositifs capteurs. Finalement, elle ne permet pas de tirer parti de nouveaux capteurs dans l'environnement et pose donc d'importants problèmes pour la dynamique des applications.

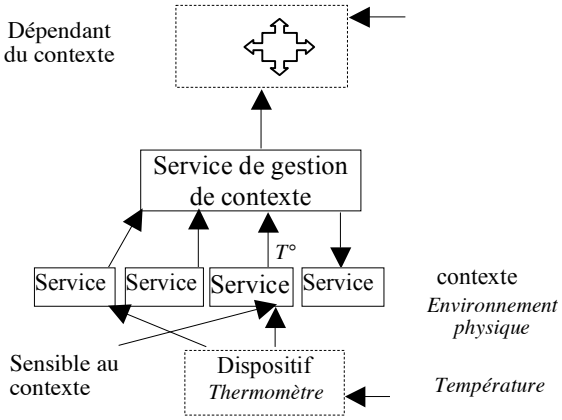


Figure 5: Service centralisé de gestion du contexte.

Dans le cadre d'une architecture basée sur les services ("Service oriented architecture"), la seconde solution consiste à développer un service de gestion de contexte [4] qui permettra de centraliser les informations et les communiquera aux applications qui les nécessitent (Figure 5). L'avantage de cette vision est de mettre en commun les informations que l'on trouve dans l'environnement; ce qui permet d'agréger facilement les informations récoltées. Par exemple, dans une maison, il est utile d'avoir un système capable de donner la température et la pression. À l'échelle d'une ville, les mêmes informations sont intéressantes mais avec un degré de précision différent. Le défaut d'un tel service de gestion de contexte est de centraliser cette gestion, ce qui est contradictoire avec la notion de contexte. Il fournit effectivement des informations sur l'environnement d'une activité, et donc des informations de contexte, mais elles ne sont pas contextuelles car elles ne dépendent plus de la tâche ou de la situation courante (telle que perçue par l'application cliente). De plus, un tel système ne permettrait une gestion de contexte efficace que dans un périmètre restreint.
La solution que nous retenons consiste à considérer que chaque dispositif impliqué dans l'architecture au dessus de la couche capteurs sera doté d'un composant de gestion de contexte dont le but est de maintenir les informations de contexte pour ses besoins propres ou pour le service des autres (Figure 6). Les dispositifs qui le désirent peuvent publier leurs capacités auprès d'annuaires permettant de les localiser, ou l'architecture peut fonctionner sur la base d'un bus auquel se connectent les dispositifs nouvellement introduits dans l'environnement. L'avantage de cette solution est, bien entendu, que de nouveaux dispositifs peuvent être dynamiquement ajoutés ou retranchés de l'environnement sans avoir à réinitialiser les autres éléments de l'architecture.

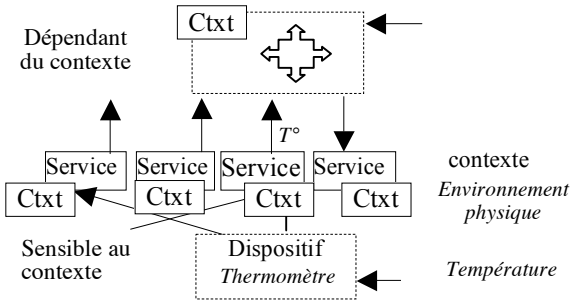


Figure 6: Le contexte, composant de chaque service et application.

La gestion de contexte sera donc implémentée comme une bibliothèque pouvant être embarquée dans n'importe quel dispositif de la figure 1. Elle devra permettre aux dispositifs actifs de demander des informations de contexte aux dispositifs (services) sensible au contexte.

### 4.2. Interaction

Les applications doivent pouvoir demander les éléments de contexte qui les concernent et certains services doivent pouvoir servir à d'autres dispositifs des éléments de contexte, typiquement agrégés.
Il est donc nécessaire de proposer un protocole permettant à tout dispositif de tirer le meilleur parti des services disponibles. On veut identifier un service, savoir quelles informations de contexte il fournit et s'adresser à lui pour obtenir cette information. Pour cela, le composant de gestion de contexte devra implémenter quelques primiti-

ves (le premier élément est la requête, le second le type de la réponse):
Id() → URI : l'identificateur du service fourni ;
Cl(URI) → URI : la classe du service identifié ;
Desc(URI) → OWL: la description de l'information que le composant peut fournir ;
Req(RDQL) → liaisons RDF : les tuples des valeurs de variables qui satisfont la requête.

Ces primitives permettent d'abord d'identifier les différents dispositifs présents dans l'environnement (idéalement à l'aide d'un protocole sur un réseau ad hoc de type bluetooth). L'identificateur permettra ensuite de s'adresser au dispositif identifié. Mais il peut aussi permettre de le caractériser plus finement (c'est le cas par exemple si cet identificateur est un URI qui permet de retrouver sur le réseau une description de l'objet identifié). La seconde primitive permet de caractériser l'objet identifié comme le membre d'une classe (au sens OWL) particulière. En principe, cette classe devrait être accessible à partir du réseau et trouver sa définition devrait permettre d'obtenir une description détaillée du dispositif identifié : on peut s'attendre, par exemple, à ce que les producteur de ces dispositifs publient leurs caractéristiques (et en particulier les requêtes que l'on peut leur adresser sur le réseau). Les deux premières primitives sont les deux seules primitives obligatoire. On peut noter qu'elles n'ont rien d'extravagant : elles sont déjà la base des protocoles RFID/EPC.
Cependant, pour les appareils fonctionnant en mode purement local, il est bon de pouvoir fournir sa description (ou plutôt celles des informations de contexte qu'il est possible d'obtenir). La primitive Desc permet d'obtenir cette information dans un langage du type OWL (ou vraisemblablement OWL-S). La quatrième primitive permet de poser des requêtes auxquelles le dispositif sait répondre et d'obtenir les informations nécessaires.
Ainsi, tout dispositif pourra :

1) rechercher dans son environnement des services capables de lui donner les informations de son contexte.
2) obtenir les caractéristiques des services retournés (par exemple, la précision d'une mesure) ;
3) s'adresser au service choisi pour obtenir l'information.

L'intérêt de cette approche est de ne pas avoir à modifier les dispositifs pour de nouvelles applications et de ne pas avoir à modifier les applications pour tirer parti de nouveaux dispositifs.

## 4.3. Exemple

Si l'on revient à l'appareil téléphonique, il recherchera dans l'environnement les services disponibles :
Id()
Deux services retourneront leur URI [1]:
→ http://panasonic.com/3d23455651p/127765543
→ http://legrand.com/sdh/com/jsd67898200
Le premier peut être aisément identifié sur le réseau comme étant un téléviseur de la classe http://panasonic.com/3d23455651p/. Il permet aux applications de savoir son état et de le piloter. Le second est la centrale de climatisation identifiable en l'interrogeant :
Cl(http://legrand.com/sdh/com/jsd67898200)
→ http://legrand.com/climctl/jsd678-mk2
La centrale de climatisation est capable de donner des informations de haut niveau correspondant à sa fonction (comme l'inutilité de rafraîchir une pièce inoccupée). Mais elle est aussi disposée à communiquer de l'information que ses capteurs ont agrégés. En particulier, elle est capable d'identifier que toutes les lumières sont éteintes. Par exemple,
Desc(http://legrand.com/climctl/jsd678-mk2) retournera :
→ <owl:Class
rdf:resource="http://ambient.org/light/#GlobalLight"/>,
<owl:DatatypeProperty
rdf:resource="http://ambient.org/light/#state"/>
permettant de répondre à la requête :
Req( SELECT ?s
WHERE <amb:GlobalLight amb:state ?s> )
par :
→ {< amb:off >}
Indiquant que toutes les lumières étant éteintes, l'utilisateur dort vraisemblablement, inutile donc de le réveiller.

# 5. Un modèle du contexte extensible

Si l'on peut ajouter des composants à tout moment, il n'est pas clair cependant qu'ils soient si facilement exploitables. Il n'y a pas de raison, a priori, que les composants proposés ainsi que les nouvelles applications soient réellement interopérables. En effet, chaque nouveau capteur fournira plus de précision ou des informations qui ne peuvent avoir été complètement prévus à l'origine. De même, les applications ne peuvent connaître tous les types de capteurs disponibles. On montre comment exploiter les langages de description d'ontologies pour résoudre ce problème.

## 5.1. De l'utilité des ontologies

Heureusement, les techniques de représentation de connaissance que l'on trouve transposées dans le langage OWL permettent de toujours préciser un concept ou une propriété sans avoir à remettre en cause ceux qui existaient au préalable.
Ainsi, il n'est nullement nécessaire, pour accomplir sa mission de savoir que le téléviseur est un téléviseur 16/9 avec disque dur intégré si l'application veut simplement savoir si l'utilisateur est occupé à regarder la télévision (ou s'il y a une activité dans la pièce).
Mais rien ne doit non plus empêcher le producteur de l'appareil de le déclarer ainsi. Il fera cela en caractérisant précisément dans l'ontologie des appareils de télévision, des dispositifs de distraction et des appareils consommateurs d'électricité le téléviseur produit.

[1] Tous les exemples sont imaginaires. Toutes ressemblances avec des marques et situations existantes serait surement fortuite.

Dans l'application donnée en exemple, l'important est que la classe du téléviseur soit une sous-classe de "dispositifs de distraction" pour que l'application en déduise une activité. Par contre, si l'application a pour but de trouver rapidement une chaîne pour que l'utilisateur ne manque pas son émission préférée, alors il est important de connaître les capacités de l'appareil et de l'interroger sur ce qu'il est à même de réceptionner.

Ainsi, l'utilisation d'ontologies pour caractériser les situations, permet d'accueillir de nouveaux appareils dont les possibilités ne peuvent être connues au départ et des nouvelles applications tirant parti de ces possibilités.
Les applications ont intérêt à être le plus générales possibles sur les informations dont elles ont besoin (la température de la pièce, l'activité d'un dispositif de distraction) alors que les systèmes de gestion de contexte doivent être le plus précis possible sur ce qu'ils produisent. Cela permettra aux applications les plus pointues d'en tirer le meilleur parti.

## 5.2. Des ontologies génériques

L'essentiel est de disposer d'ontologies suffisamment génériques pour pouvoir prendre sous leurs chapes les différentes notions impliquées dans les applications d'informatique diffuse : ressources, acteurs, lieux, dates, activités, permissions, etc. Il existe plusieurs ontologies de ce type (par exemple [4, 25]). On verra dans la section suivante comment réconcilier ces différentes ontologies.
Ces ontologies sont en général peu profondes car elles ont été conçues pour faire fonctionner la machinerie des applications d'informatique diffuse. Pour chacun des domaines, il est nécessaire de développer une description plus précise des informations exploitables. Ainsi, dans [13], nous avons proposé une description sémantique de quatre modèles différents (sémantique, géométrique, théorie des graphes, et modèles ensemblistes) pour représenter des informations de localisation. Ces informations peuvent être représentées selon l'un ou l'autre modèle, et le système saura trouver les correspondances entre les différents modèles. Nous développons aussi des modèles des propriétés physiques de l'environnement accessibles aux applications.
De tels modèles peuvent être utilisés par les applications pour caractériser les informations qui leur sont nécessaires. Les dispositifs, pour leur part, utiliseront les caractérisations les plus précises possibles pour pouvoir être exploité par les applications.

## 5.3. Exemple

Poursuivons l'exemple précédent en supposant que l'application désire connaître la température dans la pièce. L'ontologie de haut niveau lui permet de caractériser son besoin : la température est une propriété physique de la pièce, cette propriété peut être obtenue par un capteur de type ThermServer, comme un thermomètre, situé dans la même pièce. L'information nécessaire est exprimée par le graphe de la figure 7. Ce graphe correspond à un ensemble de triplets RDF (et ainsi à une requête poser à l'environnement).

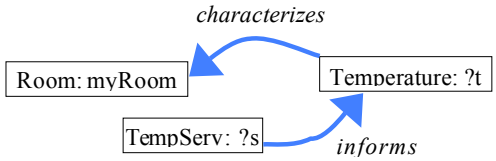


Figure 7 : Le graphe requête correspondant à l'information nécessaire à une application qui recherche dans l'environnement la température de la pièce.

La figure 8 présente l'information utilisée pour trouver un dispositif capable de donner l'information de contexte correspondant à la température de la pièce. La température elle-même sera obtenue avec le protocole présenté précédemment. Comme on peut le constater, les informations sont décrites beaucoup plus précisément que dans la requête. La pièce (concept provenant d'un modèle sémantique de localisation [13]) est devenue une chambre d'hôtel. Quant au capteur de température, il est la centrale de climatisation qui, disposée à fournir cette information, est décrite comme un serveur de température (elle peut, bien entendu, être décrite comme beaucoup d'autre choses).

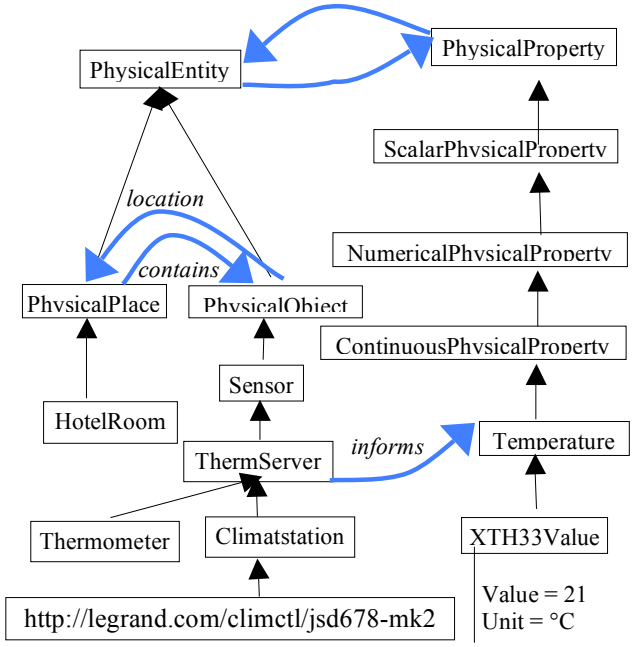


Figure 8 : Le raffinement et l'instanciation de l'ontologie qui entre effectivement en correspondance avec les besoins de l'application exprimés à la figure 7.

# 6. Exploiter des ressources hétérogènes

Le système de gestion de contexte proposé permet d'introduire dans l'environnement de nouveaux dispositifs en étendant l'ontologie de telle sorte que les anciennes applications puissent en tirer parti. Ceci fonctionne si les acteurs ont pour référence les mêmes bases ontologiques. Il n'y a pas vraiment de raison pour cela. Au contraire, deux producteurs de capteurs peuvent avoir de légitimes raisons d'utiliser des ontologies différentes pour décrire

leurs composants. Il n'est pas concevable de supposer une ontologie unique dans laquelle tout est intégré.
Ceci entraîne un problème lorsqu'il s'agit d'apparier les contextes produits et les informations de contexte nécessaires à une application. Trois approches sont possibles pour assurer l'interopérabilité dans les environnements d'informatique diffuse :

— La standardisation a priori des ontologies utilisées : comme indiqué plus haut, ceci est difficile à obtenir, par ailleurs ce n'est pas souhaitable car cela mettrait un frein à l'évolution nécessaire des ontologies et de la technologie ;
— La maintenance de médiateurs entre ontologies : que ce soit sur un mode distribué ou centralisé on peut imaginer que les correspondances entre ontologies puissent être librement mises à disposition (que ce soit par les producteurs de capteurs, les développeurs d'application ou les utilisateurs eux-mêmes) et que les applications puissent les exploiter si elles doivent mettre en correspondance deux ontologies ;
— Un service d'alignement d'ontologie : il permet lorsque le besoin s'en fait sentir de trouver la correspondance entre deux ontologies.

Ces trois solutions ne sont pas incompatibles et peuvent fonctionner de concert. S'accorder sur les ontologies de haut niveau est souhaitable et possible et laisser les niveaux plus spécifique se développer librement permet de prendre en compte rapidement l'évolution d'un domaine.

## 6.1. Un service d'alignement d'ontologies

Nous préconisons la mise en œuvre d'un (ou de plusieurs) service(s) d'alignement d'ontologies. Le rôle d'un tel service est d'aider des agents (en l'occurrence ici les gestionnaires de contexte) à trouver les correspondances entre les différentes ontologies auxquelles ils sont confrontés. Ces services sont à même d'offrir les opérations suivantes :

— Trouver une ontologie proche d'une ontologie particulière ;
— Stocker (et retrouver) des alignements déjà obtenus entre différentes ontologies ;
— Calculer dynamiquement la correspondance entre deux ontologies ;
— Traduire les requêtes et réponses aux requêtes vers des gestionnaires de contextes utilisant des ontologies différentes.

De même que les ontologies, les services d'alignement devront être accessibles aux applications et aux gestionnaires de contexte par le biais du réseau. Les services d'alignement utilisent et adaptent à leurs propres besoins une interface fonctionnelle permettant de manipuler explicitement les alignements entre ontologies développées dans le contexte du web sémantique [12].

## 6.2. Exemple

Considérons le même exemple que précédemment avec une centrale de climatisation utilisant une ontologie particulière du domaine de la climatisation (http://clim.org/ontologies/devices.owl/v1.1) et l'appareil téléphonique utilisant une ontologie modélisant les situations dans lesquelles se trouve un humain (http://psycho.illinois.edu/~andersen/attitudes/onto.rdf).
On comprend bien pourquoi, a priori, ces deux modèles du monde ont peu de raison d'être les mêmes.
Ceci dit, notre téléphone a juste besoin de savoir si une personne est trop occupée pour pouvoir la déranger… et comme on trouve des climatisations en de nombreux endroits et des appareils téléphoniques dans presque toutes les poches, il y a de fortes chances que ces deux ontologies aient déjà été mises en correspondance.
Cette fois-ci, l'appareil téléphonique va pouvoir s'adresser au service d'alignement en ces termes :
Align( http://clim.org/ontologies/devices.owl/v1.1,
http://psycho.illinois.edu/~andersen/attitudes/onto.rdf)
→ al23889
À noter qu'ici, le service répond directement par un alignement sans que l'on sache s'il a été calculé ou stocké. Le protocole permet de spécifier les paramètres, le type d'algorithme à utiliser ou s'il faut chercher en priorité l'alignement dans la bibliothèque. La réponse retournée n'est pas l'alignement mais un identificateur permettant au service de le retrouver si nécessaire. L'application peut demander à traduire la requête à poser au gestionnaire de contexte de la centrale de climatisation.
Translate(al24889,
"SELECT ?s WHERE <psy:room psy:lighting ?s>")
→ SELECT ?s WHERE <amb:GlobalLight amb:state ?s>
Il pourra poser la requête retournée à la centrale et demander l'interprétation du résultat :
Translate(inv(al24889), {< amb:off >})
→ {< psy:dark >}

## 7. Travaux connexes

L'importance de l'appréhension du contexte en informatique diffuse est largement reconnue. Aussi, comme nous l'avons présenté, il existe de nombreux modèles de gestion de contexte. Un modèles proche de celui présenté ici est celui des contextors [5] qui propose une bibliothèque d'éléments capables de fournir les informations de contexte et permet de combiner les informations contextuelles distribuées. Par contre, ce dernier modèle ne considère pas l'aspect dynamique : l'ajout d'éléments sans interrompre le fonctionnement des dispositifs.
Sur le plan de l'utilisation des technologies du web sémantique dans la représentation du contexte, on a vu qu'il existe plusieurs propositions pour étendre les langages du web sémantique afin de contextualiser les assertions [2, 15, 18]. Les applications telles que décrites ici auraient tout intérêt à utiliser l'une de ces propositions. En ce qui concerne l'utilisation de OWL pour représenter concrètement l'information de contexte, [25] propose une ontologie de haut niveau de l'information contextuelle pour l'informatique diffuse. L'intérêt de l'approche est la même que celle présentée au §5: pouvoir étendre l'ontologie. Les auteurs ne proposent cependant aucune piste pour étendre celle-ci dynamiquement. [1] développe pour sa part une ontologie contextuelle en OWL où le contexte est composé des centres d'intérêt d'un utilisateur

permettant au système d'adapter la présentation de pages web. Il ne s'agit pas d'informatique diffuse au sens où toutes les interactions sont centralisées dans un serveur et l'application n'interagit jamais avec un environnement dynamique ou inconnu.

## 8. Discussion et conclusion

Notre objectif est de considérer l'aspect dynamique de la gestion de contexte. Ce problème a été abordé en proposant une architecture de composants distribués (proche des contextors) utilisant les technologies du web sémantique. Les composants permettent d'ajouter à tout moment de nouveaux éléments capables de fournir des informations sur le contexte d'une application. L'utilisation de RDF et de OWL permet d'assurer l'interopérabilité entre des composants développés indépendamment, d'une part en utilisant les propriétés d'ouverture inhérentes à ces langages et d'autre part en exploitant les travaux d'alignement de modèles.

Il n'aura pas échappé au lecteur que l'approche présentée ici suppose l'adhésion du monde entier à quelques bases : RDF, OWL, un protocole d'identification (qui pourrait être celui esquissé ici mais aussi un protocole du type EPCIS utilisé avec les identificateurs radio). D'une part, ces technologies sont déjà recommandées et utilisées et ne représentent pas une implication disproportionnée de la part des industriels. D'autre part, il nous semble cependant que c'est la condition sine qua non pour des applications d'informatique diffuse viables.

Nous travaillons actuellement au développement de démonstrateurs de cette technologie : nous disposons d'une version centralisée exploitant les ontologies décrites en OWL et permettant l'interaction avec les capteurs. L'alignement d'ontologies est disponible indépendamment et reste à intégrer à ce système.

## Bibliographie


[1] F. Abel, J. Brase, Using semantic web technologies for context-aware information providing to mobile devices, Technical Report, L3S, Hannover (DE), 2004.

[2] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, H. Stuckenschmidt, C-OWL : contextualizing ontologies, Proc. 2nd ISWC, Sanibel Island (FL US), *LNCS* 2870:164-179, 2003

[3] M. Chalmers, A Historical View of Context, *Computer supported cooperative work* 13(3):223-247, 2004.

[4] H. Chen, T. Finin, A. Joshi, An Ontology for Context-Aware Pervasive Computing Environments, *Knowledge engineering review* 18(3):197-207, 2004.

[5] J. Crowley, J. Coutaz, G. Rey, P. Reignier, Perceptual components for context aware computing, Proc. International Conference on Ubiquitous Computing, Göteborg (SW), pp117-134, 2002.

[6] J. Coutaz, J. Crowley, S. Dobson, D. Garlan, Context is key, *Communications of the ACM* 48(3):49-53, 2005.

[7] M. Dean, G. Schreiber Eds, OWL Web Ontology Language: Reference, W3C Recommendation, 2004.

[8] J. de Kleer, An assumption-based TMS, *Artificial Intelligence* 28(2):127-162, 1986.

[9] A. Dey, D. Salber, G. Abowd, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Human-Computer Interaction* 16:97-166., 2001

[10] A. Dey, Understanding and using context, *Personal and ubiquitous computing* 5(1):4-7, 2001.

[11] P. Dourish, Seeking a foundation for context-aware computing, *Human-Computer Interaction*, 16(2-3), 2001.

[12] J. Euzenat, An API for ontology alignment, Proc. 3rd ISWC, Hiroshima (JP), *LNCS* 3298:698-712, 2004.

[13] T. Flury, G. Privat, F. Ramparany, OWL-based location ontology for context-aware services, Proc. Artificial Intelligence in Mobile Systems, Nottingham (UK), pp. 52-58, 2004.

[14] R. Guha, Contexts : a formalization and some applications, PhD thesis, Stanford university (CA US), 1991 (Technical Report STAN-CS-91-1399-Thesis et MCC ACT-CYC-423-91).

[15] R. Guha, R. Fikes, R. McCool, Contexts for the Semantic Web, Proc. 3rd ISWC, Hiroshima (JP), *LNCS* 3298:32-46, 2004.

[16] K. Henricksen, J. Indulska, A. Rakotonirainy, Modelling context information in pervasive computing systems, Proc. 1st International Conference on Pervasive Computing, pp167-180, 2002.

[17] H. Hu, D.-L. Lee, Semantic Location Modelling for Location Navigation in Mobile Environment, Proc. 5th IEEE International Conference on Mobile Data Management, pp52-61, 2004.

[18] O. Khriyenko, V. Terziyan, Context description framework for the semantic web, submitted.

[19] G. Klyne, J. Carroll, Eds., Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, 2004

[20] J. McCarthy, Notes on formalizing context, Proc. 13th IJCAI, Chambéry (FR), pp555-560, 1993.

[21] A. Narayanan, Realms and States: A framework for context aware mobile computing, Proc. 1st international workshop on mobile commerce, Rome (IT), pp48–54, 2001.

[22] B. Schilit, N. Adams, R. Want, Context-Aware Computing Applications, Proc. IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz (CA US), 1994.

[23] A. Seaborne, RDQL — A Query Language for RDF, W3C Member submission, 2004.

[24] L. Serafini, P. Bouquet, Comparing formal theories of context in AI, *Artificial Intelligence* 155:1-67, 2004.

[25] X. H. Wang, D. Q. Zhang, T. Gu, H. Keng Pung, Ontology based context modeling and reasoning in OWL, Proc. 2nd PerCom workshop on Context Modeling and Reasoning (CoMoRea), Orlando (FL US), 2004.