Filtrage contextuel par cache pour application de réalité augmentée mobile

Bertrand Richard, Elöd Egyed-Zsigmond, Sylvie Calabretto

Université de Lyon
LIRIS UMR 5205, INSA de Lyon
20, avenue Albert Einstein
F-69621 Villeurbanne Cedex

prenom.nom@liris.cnrs.fr

RÉSUMÉ. Le développement actuel des applications mobiles nécessite de plus en plus la prise en compte du contexte de l'utilisateur pour permettre leur optimisation. Dans cet article, nous proposons une approche de filtrage contextuel d'information pour les utilisateurs nomades exploitant cinq dimensions de contexte : spatial, temporel, technique, social et activité. Nous présentons les deux méthodes de filtrage que nous avons testées ainsi que le résultat de nos expérimentations préliminaires.

ABSTRACT. The current development of mobile applications requires more and more to take into account the context of the user, to allow optimizations. In this paper, we propose a contextual information filtering approach for mobile users, involving five context dimensions: spatial, temporal, technical, social and activity. We present the two filtering methods we tested and the results of our preliminary experiments.

MOTS-CLÉS : contexte, cache, système d'informations, requêtes contextuelles.

KEYWORDS: context, caching, information systems, contextual queries.

DOI:10.3166/DN.15.1.57-77

Document numérique – n° 1/2012, 57-77

1. Introduction

L'émergence des technologies mobiles permet aujourd'hui aux utilisateurs nomades d'accéder quasiment n'importe où et n'importe quand à un ensemble important d'informations. Cette recherche d'information mobile offre de nouveaux défis, sur le plan de la conception des requêtes, de la consultation des résultats limitée par le temps et l'espace et de la précision de ces résultats.

Le projet Vietaville est un projet « serious game » soutenu par la région Rhône-Alpes et basé sur la collaboration entre le laboratoire LIRIS et l'entreprise Les Tanukis. Dans ce cadre, nous nous efforçons de fournir aux utilisateurs nomades de manière proactive les informations environnantes, déposées par d'autres utilisateurs, qui sont susceptibles de les intéresser, pour les visualiser par le biais d'une application de réalité augmentée mobile. Les requêtes de recherche d'information dans notre situation ne sont donc pas formulées directement par l'utilisateur, mais construites automatiquement à partir de son contexte et de ses préférences. Les informations proposées par l'application peuvent être de type « tags muraux », informations horaires, menus de restaurants, etc. Il est donc nécessaire pour atteindre notre but de formaliser les informations du contexte de l'utilisateur et de mettre en place un système permettant de filtrer l'ensemble des informations disponibles pour pouvoir fournir très rapidement à l'utilisateur celles qui seront pertinentes dans son contexte.

Après un état de l'art sur la recherche d'information contextuelle mobile, nous présentons dans cet article le contexte utilisateur que nous prenons en compte pour cette recherche d'information, puis notre principe de filtrage contextuel pour obtenir des résultats précis pertinents en un temps minimum. Nous présentons également les deux méthodes de filtrage que nous avons testées, ainsi que le système de cache contextuel que nous avons mis en place pour améliorer la résolution de requêtes proches. Enfin, nous présentons les résultats d'expérimentations préliminaires effectuées sur notre système de tests.

2. Contexte et recherche d'information pour réalité augmentée mobile

2.1. Contexte et modèles de contexte

Le contexte est un concept concernant de nombreux domaines (Guha, 2003). Notre définition générale du contexte est : « l'ensemble des circonstances entourant une situation donnée ». D'un point de vue informatique, le contexte peut être interprété de différentes façons, suivant le point de vue que l'on choisit d'adopter ; il peut s'agir du contexte de l'application, de l'utilisateur ou d'une autre entité d'intérêt.

Du point de vue de l'application, le contexte englobe la situation physique dans laquelle l'application s'exécute. Cela inclut le système d'exploitation, la machine physique sur laquelle elle s'exécute, et les différents outils à disposition, réseau, autres applications, etc.

Du point de vue de l'utilisateur, ce contexte concerne « son état physique, émotionnel, social et informationnel » (Dey 1999 ; Dey 2001). Pour Zetie (2002), cette notion englobe également « les connaissances sur sa tâche, ses intentions, ses buts, son historique et ses préférences », et sa modélisation et son utilisation ont pour but d'améliorer les performances d'une application. De manière plus générale, concernant une entité quelconque, Funk et Miller (1997) décrivent le contexte comme « tout ce qui entoure un objet d'intérêt, incluant l'état d'esprit de tout humain concerné ». De nombreuses catégorisations du contexte, comme celle de Bradley (2005) présentée sur la figure 1, ont déjà été proposées, organisant ses composantes selon différentes dimensions.

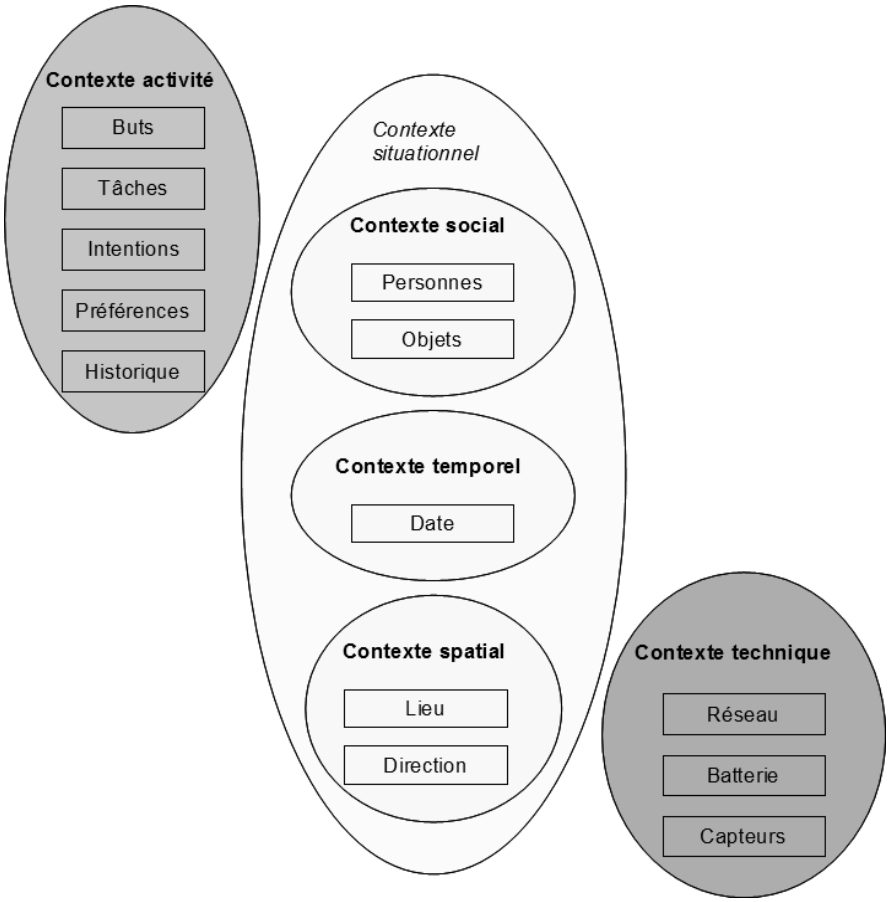

Figure 1. Modèle de contexte informatique basé sur le modèle de Bradley et Dunlop, incluant le contexte technique, le contexte de l'activité, le contexte temporel, le contexte spatial et le contexte social

Les dimensions les plus couramment rencontrées dans les modèles de contexte (Bolchini *et al.*, 2007) sont les cinq dimensions suivantes :

– *le contexte technique* qui regroupe les caractéristiques techniques du dispositif utilisé, les différents capteurs du téléphone (caméras, micros, GPS, orientation, mouvement), la taille de l'écran, le réseau disponible, le système d'exploitation, les différentes technologies à disposition, etc. Ce contexte permet principalement de déterminer quelles informations sont à portée, et comment elles peuvent être utilisées. Il définit les limitations techniques de l'appareil ;

– *le contexte spatial* qui consiste en la localisation et l'orientation du dispositif, plus toutes informations utiles sur le lieu où se trouve l'utilisateur. Ce contexte permet de déterminer les informations pertinentes par rapport au lieu de l'utilisateur, comme par exemple, la ville où il se trouve ;

– *le contexte temporel* qui permet de déterminer d'une part, si une information est pertinente par rapport à sa date d'émission et sa date de réception, et concerne également l'historique de l'utilisateur. Ce contexte permet donc principalement de déterminer la validité temporelle d'une information, mais aussi de prendre en compte certains aspects comme les fréquences d'utilisation, de demande d'information, etc. Cette dimension permet également d'utiliser le contexte comme une sorte de trace, puisque ses éléments sont temporellement situés ;

– *le contexte social* qui est composé de toutes les interactions avec d'autres entités humaines, ou représentants de ces entités. On trouve ainsi dans ce contexte les amis de l'utilisateur (au sens Facebook), ses proches, ses différentes connaissances, ses collègues, ses voisins… Cette dimension du contexte permet de mettre en relation des informations supplémentaires basées sur les interactions entre utilisateurs ;

– *le contexte de l'activité* (Kaenampornpan, 2004), plus difficile à définir, puisqu'il concerne directement les tâches et intentions de l'utilisateur. Ce contexte comprend entre autres, la tâche courante de l'utilisateur, ses buts à court et long termes, ses tâches secondaires… Cette dimension permet de donner la priorité aux informations utiles pour l'utilisateur dans sa tâche. Il s'agira par exemple dans le cas d'un utilisateur de VieTaVille de grandes catégories hiérarchisées d'activité, comme « travail », « restauration », « loisir », « cinéma », etc., permettant de définir le type d'information que l'utilisateur est susceptible de rechercher.

2.2. Recherche d'information contextuelle

La recherche d'information est devenue une activité pratiquée couramment par toute personne se connectant à Internet, incitant à proposer de meilleures solutions, plus efficaces et plus pertinentes pour les systèmes de recherche d'information (Boughanem, 2008). La prise en compte des informations du contexte de l'utilisateur est une des pistes suivies pour fournir des résultats de recherche plus adaptés à celui-ci. Ainsi, faire de la recherche d'information contextuelle signifie souvent adapter une requête ou les résultats d'une requête en fonction du contexte de l'utilisateur, calculé au moment de la requête. Ce contexte contient généralement un

historique des actions de l'utilisateur permettant de déterminer les objectifs de sa requête. Cette pratique de la recherche contextuelle implique différentes étapes, allant de la modélisation du contexte jusqu'à la définition de la pertinence cognitive et la modélisation des interactions entre un utilisateur et un système de recherche d'information.

Dans le cas de notre étude, il n'y a pas de requêtes explicites. Nous sélectionnons des informations contextualisées correspondant à un contexte utilisateur donné. En effet, nous voulons proposer du contenu pertinent pour l'utilisateur sans que lui-même ait formulé une requête particulière. À chaque modification de contexte, la sélection doit être mise à jour. Le filtrage des informations pertinentes pour l'utilisateur est mené de manière « semi-continue » et son optimisation est donc primordiale, principalement pour les deux dimensions qui varient le plus : le temps et l'espace.

De nombreux systèmes exploitent actuellement le potentiel contextuel pour la recherche d'information (Baldauf *et al.*, 2007). De plus, des systèmes de filtrage d'information basés sur l'utilisation du contexte (Kirsch-Pineiro, 2005) sont également étudiés, pour permettre des réponses plus adaptées aux utilisateurs. L'évaluation de tels systèmes est assez compliquée, comme le relève (Tamine-Lechani *et al.*, 2009), et diffère fortement de l'évaluation des systèmes de recherche d'information classiques. L'utilisation de cache de résultats de requêtes est également un point déjà abordé dans la littérature, notamment dans les travaux de (Chaudhuri *et al.*, 1995) et plus récemment (Miranker *et al.*, 2002) qui propose l'utilisation d'un cache de requêtes par approximation pour optimiser la recherche de données dans des bases hétérogènes.

2.3. Recherche mobile d'information

Dans le cas d'applications pour mobiles, plusieurs contraintes contextuelles particulières sont à prendre en compte (Wigelius, 2009). En effet, les dispositifs mobiles bien que de plus en plus performants, disposent de capacités limitées, tant au niveau du stockage, que de l'autonomie et de la puissance de calcul. Le type d'application sur lequel nous travaillons demande des temps de réponse rapides pour éviter que l'utilisateur attende son information. Il est donc nécessaire en amont de bien filtrer le volume d'informations pour ne garder que les plus pertinentes, mais également de ne laisser passer que celles que l'utilisateur pourra effectivement utiliser avec son mobile en adaptant éventuellement l'information aux performances offertes (Lemlouma, 2004 ; Weissenberg, 2004 ; Korpipää, 2004). Le problème des requêtes géographiques est un domaine particulièrement important dans la recherche d'information, en constante évolution du fait, principalement, du développement permanent des réseaux mobiles (Ilarri, 2010). Les applications utilisant partiellement le contexte ou basées sur la recherche d'information contextuelle sont de plus en plus fréquentes actuellement sur mobile. Nous pouvons citer comme principaux exemples Sitelens (White, 2009), Layar, Wikitude... Ces applications, qui ont pour but de fournir à l'utilisateur des informations sur ce qui l'entoure, quel que soit l'endroit où il se trouve, ont besoin d'obtenir des données ciblées, dans un temps de

réponse très court et de façon proactive (Ajanki, 2010). Des études (Kumar, 2005 ; Gronau, 2003) montrent l'efficacité du raisonnement à partir de cas basés sur le contexte pour fournir des résultats proches intéressants pour l'utilisateur, en adaptant sa requête en fonction de requêtes similaires ayant renvoyé des résultats satisfaisants.

Une autre façon de procéder pour optimiser les résultats est de recourir au retour utilisateur, explicite ou implicite (Martins, 2009), une fois les informations présentées, pour améliorer les futures requêtes similaires en pratiquant un apprentissage et en pondérant les informations.

2.4. Cache sémantique

Le principe du cache contextuel se rapproche de l'utilisation du cache sémantique (Ren, 2000). En effet, comme pour le cache sémantique, il est défini par des couples requêtes/résultats, qui permettent de réutiliser tout ou partie des résultats de précédentes requêtes si les paramètres de celles-ci sont proches ou incluent les paramètres de la nouvelle requête. Chaque résultat de requête est donc associé à une « région sémantique » pour laquelle il est valide, et qui sera comparée à la région sémantique des nouvelles requêtes arrivant sur le système. L'efficacité d'un tel cache est alors déterminée par la proximité sémantique des requêtes (Ren, 2003).

L'utilisation de cache sémantique est très dépendante du type de données et de requêtes manipulées, et doit être pensée intelligemment pour optimiser la réutilisation des données (Arens, 1994). Dans le cas de données spatiales, ce qui constitue une partie de notre contexte, les méthodes utilisées ne sont pas les mêmes suivant qu'il s'agit d'utilisateurs fixes ou mobiles, et de données fixes ou mobiles. Ainsi, certaines méthodes telles que l'utilisation de diagrammes de Voronoi (Zheng, 2001) pour déterminer la région de validité de l'information peut être problématique dans le cas d'objets mobiles.

Enfin, l'utilisation de cache sémantique pour gérer des données contextuelles implique quelques complications pour maintenir la validité et la cohérence du cache. Celles-ci dépendent de chaque dimension du contexte et se posent alors des problèmes de validité des données spatiales, temporelles, techniques (Xu, 1999), ainsi que des choix pour la mise à jour et le remplacement de ces caches (Zengh, 2002).

3. Présentation du projet Vietaville

Vietaville est un projet pour mobile qui permet aux habitants de personnaliser virtuellement les murs de leur ville en déposant des textes, des citations, des poèmes, des images ou des photos dans l'environnement urbain (première partie du projet) puis sur les murs (deuxième partie du projet), d'abord par le biais d'une application Facebook reliée aux API Google Map, et dans un second temps à travers une application de réalité augmentée pour téléphone mobile.

La première partie du projet consiste en une application web permettant de déposer et visualiser des « Informations » géolocalisées sur une carte (figure 2).

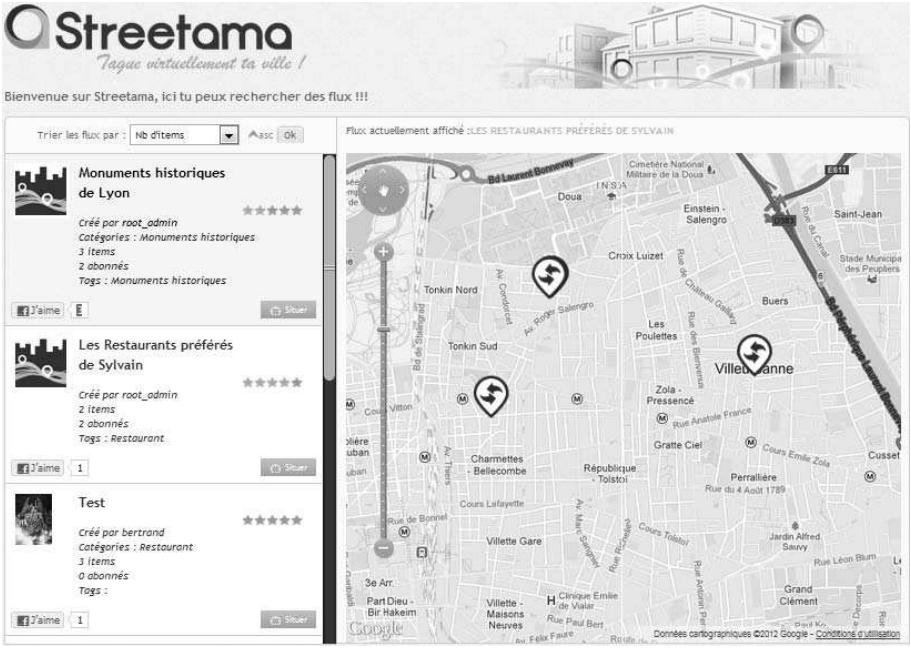

Figure 2. Capture d'écran du site Streetama.net, façade web du projet Vietaville. À gauche, une liste des flux à portée, à droite leur localisation sur une carte

Chaque utilisateur possède un ou plusieurs flux d'informations. Ces flux, ainsi que les informations qu'ils contiennent sont contextualisés (géolocalisés, catégorisés...). De plus, chaque utilisateur peut s'abonner à des flux pour bénéficier des informations qu'ils contiennent, et les visualiser sur son dispositif mobile, sur une carte (figure 3) ou en réalité augmentée (figure 4) pour la seconde partie du projet.

4. Problématique

Les problèmes de recherche d'information (temps de réponse, pertinence des résultats...) sont encore plus critiques en contexte mobile. Lorsqu'il s'agit de récupérer des informations de type image pour les utiliser dans un cadre de réalité augmentée, il paraît important de minimiser le temps de réponse aux requêtes contextuelles. Nous nous intéressons dans cet article aux moyens d'améliorer cet aspect critique, en proposant l'utilisation d'un système de filtrage contextuel et de cache de requêtes contextuelles. Nous présentons les deux méthodes que nous avons mises en place pour permettre un filtrage contextuel morcelé afin d'obtenir une plus

grande réutilisabilité des résultats de ces filtrages, accélérant ainsi les temps de réponse aux requêtes proches ultérieures.

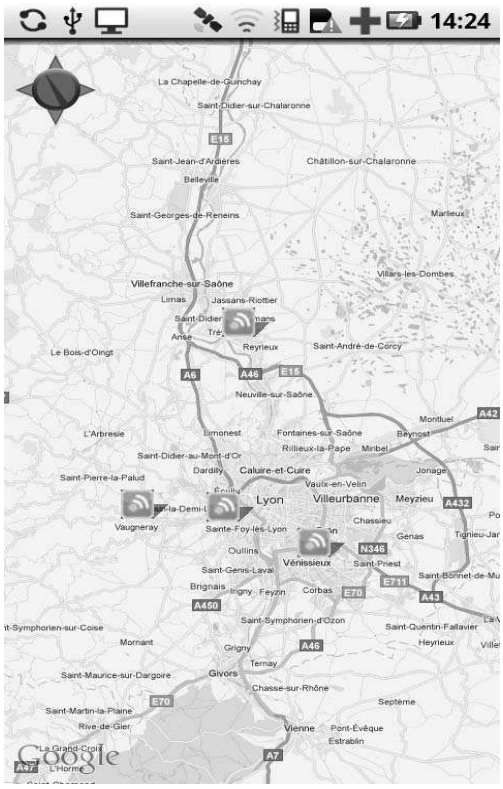

Figure 3. Application mobile Streetama : visualisation des objets des flux à portée sur une carte

5. Nos propositions de filtrage contextuel

Dans le cadre de notre projet Vietaville, nous disposons d'une base de flux d'informations contextualisées présentée à la figure 5. Le modèle de contexte que nous avons choisi d'utiliser regroupe les cinq dimensions mises en évidence dans la partie précédente, adaptées aux spécificités des mobiles. Le mode de fonctionnement de l'application est le suivant : les utilisateurs consultent leur téléphone portable pour visualiser en temps réel et en réalité augmentée des informations géolocalisées autour d'eux. À leur connexion sur l'application, une requête contextuelle est envoyée à un serveur, qui leur transmet les informations pertinentes parmi un ensemble d'informations réparties dans des flux.

Figure 4. Application mobile Streetama : visualisation en réalité augmentée d'un objet, logo Insa, positionné sur un bâtiment

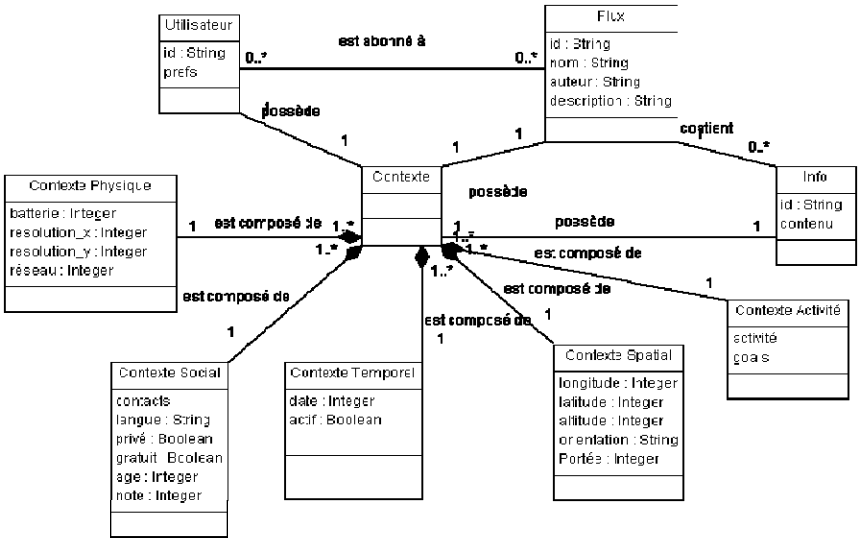

Figure 5. Modèle de données de Vietaville. Un flux contient des informations, et chaque flux et information sont contextualisés suivant cinq dimensions

Notre but est de fournir les informations correspondant à une requête virtuelle R_{user} de l'utilisateur, requête composée d'un contexte M_{user}, et de contraintes C_{user} sur les éléments de ce contexte.

$$R_{user} = (M_{user}, C_{user})$$

Nous présentons M_{user} et C_{user} en détail à la fin de cette formalisation.

L'ensemble des données à filtrer par cette requête est un ensemble de flux F tel que $F = \{ f_p | p \in [1..|F|] \}$.

Chaque flux f est défini comme un couple formé d'un ensemble I d'informations i et d'un ensemble M_f de métadonnées contextuelles, divisible en cinq dimensions :

$$f = (M_f, I_f)$$

$$I = \{ i_q | q \in [1..|I|] \}$$

$$M_f = \{ M_{temp}, M_{tech}, M_{soc}, M_{act}, M_{spa} \}$$ où Mtemp regroupe les données temporelles, M_{tech} les données techniques (état de la batterie, résolution écran, périphériques disponibles...), M_{soc} les données sociales (flux amis, listes noires...), M_{act} les données d'activité (agenda) et M_{spa} les données spatiales (coordonnées GPS).

Ainsi, un flux d'informations basé sur le campus à Lyon et prodiguant des informations sur les événements s'y déroulant pourra avoir un contexte tel que :

– pour M_{temp} : flux actif, mis en place le 01/01/2011 à 08h00, pas de fin de validité ;

– pour M_{tech} : informations disponibles par défaut à une résolution de 320*320, nécessitant un débit minimal de 100 kbit/s pour une utilisation optimale, ainsi qu'au moins 10 % de la batterie ;

– pour M_{soc} : ce flux est disponible pour tous, gratuitement, en français et en anglais, sans limite d'âge ;

– pour M_{act} : c'est un flux proposant les menus du jour des restaurants universitaires du campus, classé dans les catégories d'activité « restauration » et « information » ;

– pour M_{spa} : il est localisé en (45.783028, 4.881148) avec une portée de 500 m.

Chaque information i est elle-même un couple constitué d'un contenu et d'un ensemble M de métadonnées contextuelles.

$$i = (\gamma_i, M_i)$$

Chaque sous-ensemble de métadonnées peut être défini comme un ensemble de couples attributs/valeurs (n,v) éventuellement définis dans une ontologie O. Cette dernière peut par exemple permettre de définir des catégories d'activité en fonction des éléments détectés dans l'agenda d'un utilisateur.

$$M_{temp} = \{(n, v) | n \in O_{temp} \wedge v \in O_{temp}\}$$

La requête de filtrage R_{user} émise par l'utilisateur et présenté au début de cette section est composée d'un contexte M_{user}, et de contraintes C_{user} sur les éléments de ce contexte.

$$R_{user} = (M_{user}, C_{user})$$

L'ensemble de contraintes C_{user} de l'utilisateur est une conjonction de contraintes c_i simples

$$C_{user} = \{c_1 \wedge ... \wedge c_n\}$$

chaque contrainte simple étant de la forme suivante, où v_n représente la valeur de l'attribut n dans le contexte de chaque flux M_f ou information M_i, v_c la valeur contrainte, pouvant ou non provenir du contexte utilisateur et o l'opérateur de comparaison.

$$c_i = v_n \, o \, v_c, o \in \{<, >, =, \leq, \geq, \neq, \in, \notin\}$$

Plusieurs contraintes de l'ensemble C_{user} peuvent s'appliquer au même attribut n, notamment pour contraindre le contexte temporel dans un intervalle donné.

On observe une exception à cette formalisation, qui porte sur la contrainte géographique et qui implique un calcul de distance géographique entre le couples latitude/longitude de M_f ou M_i et celui de M_{user} exprimée comme suit:

$$c_{géo} = dist(v_{lat,f}, v_{lon,f}, v_{lat,u}, v_{lon,u}) \, o \, v_c, o \in \{<, >, =, \leq, \geq\}$$

où v_{lat} est la valeur de l'attribut représentant la latitude, v_{lon} celui exprimant la longitude, $v_{lon,f}$ est la valeur de la longitude dans le contexte du flux f et $v_{lon,u}$ la valeur de la longitude dans le contexte de l'utilisateur u.

Par exemple, la requête R_{user} d'un utilisateur contiendra entre autres ses coordonnées géographiques (longitude $v_{lon,u}$, latitude $v_{lat,u}$, altitude $v_{alt,u}$) dans l'ensemble M_{spa} et une contrainte sur celles-ci indiquant qu'il ne désire que les informations situées à moins d'un kilomètre de sa position ($c_{géo}$ = dist($v_{lat,f}$, $v_{lon,f}$, $v_{lat,u}$, $v_{lon,u}$)<1000), ainsi qu'une contrainte sociale limitant les flux à ceux sur lesquels ses amis ont déjà donné un avis. Les flux sont des ensembles d'informations émanant d'une même entité considérée comme le propriétaire du flux. On pourra donc avoir des flux d'informations appartenant à des magasins ou restaurants qui désirent présenter virtuellement leurs articles et tarifs, mais également des flux plus communautaires, comme les informations sur les monuments importants (historiques, administratifs...) des villes, etc. Les informations composant ces flux seront de natures différentes, comme des tags sur les murs des bâtiments ou des vidéos intégrées virtuellement à l'espace réel (on peut imaginer un serveur de restaurant venant présenter chaque plat du menu devant son enseigne).

Il nous faut donc construire une fonction de filtrage permettant de réduire à moindre coût l'ensemble I à un ensemble I', tel que chaque élément de I' satisfasse aux contraintes contextuelles du filtre de la requête (portant sur M_{user}).

5.1. Utilisation de cache de requêtes

Dans les deux méthodes proposées, un cache des résultats de requêtes est mis en place afin d'améliorer les performances de l'application. L'utilisation de ce cache est soumise à un calcul de distance entre les requêtes. Cette distance entre requêtes correspond à l'ensemble pondéré des distances pour chaque dimension contextuelle (mesure de distance physique pour la dimension spatiale, mesure sémantique pour l'activité, etc.).

Ainsi, lorsqu'une requête arrive sur le système, la première action est de déterminer si une requête ayant une composante proche a été exécutée récemment. Dans le cas où le cache possède le résultat d'exécution d'une requête proche, alors celui-ci est utilisé, à la manière du raisonnement à partir de cas, où un cas proche connu est adapté pour fournir la solution à un problème.

Par exemple, dans le cas de deux touristes étrangers se déplaçant dans le centre de Lyon vers midi, qui voudraient trouver un restaurant à proximité pour se sustenter : le premier connecté à l'application Vietaville envoie une requête contextuelle au système qui lui construit la liste de tous les restaurants et cafés dans un rayon de un kilomètre autour de lui et lui permet de visualiser via son application de réalité augmentée mobile les menus traduits proposés par chacun. Le second touriste, situé à une rue de distance, émet « la même requête » dans une fenêtre temporelle proche. La requête en elle-même ne sera pas identique, puisqu'il ne se situe pas aux mêmes coordonnées géographiques que le précédent, qu'elle n'est pas émise au même moment, qu'il ne possède pas les mêmes contacts, et ne parle peut-être pas la même langue. Cependant, la distance entre les requêtes sera suffisamment faible pour que le résultat de la première, qui était mis en cache, soit réutilisé pour la seconde, améliorant nettement le temps de réponse.

L'interrogation de ce cache et la construction des requêtes est le rôle de notre *Usine à filtres*. Celle-ci reçoit les paramètres de la requête en provenance de l'utilisateur, contenant son contexte et les contraintes à vérifier, et interroge le cache pour déterminer si une partie de la nouvelle requête a déjà été traitée récemment. En fonction des éléments présents dans le cache, l'*usine* construit la nouvelle requête à appliquer à la base de données, et fournit le résultat à l'utilisateur. Les deux méthodes de recherche d'éléments pertinents dans le cache sont détaillées ci-après. La maintenance du cache fait également partie des fonctions de l'*usine à filtres*. Lorsque les données entreposées dans le cache sont trop vieilles, elles sont remplacées par le résultat de la requête en cours. La cohérence du cache par rapport aux tables sous-jacentes est maintenue par l'utilisation de « Triggers » permettant de répercuter les modifications des tables sur les données en cache.

5.2. Méthode de filtrage en cascade

La première méthode que nous avons mise en place pour filtrer les flux et les informations de notre application consiste en une liste de filtres contextuels appliqués successivement à l'ensemble des données.

Chacun de ces filtres porte sur une des dimensions du contexte telles qu'identifiées précédemment. Ainsi, l'ensemble des flux et informations pourront être filtrés, par exemple, suivant la dimension spatiale pour ne garder que les données dans un rayon proche de l'utilisateur, puis suivant la dimension temporelle pour ne garder que les données publiées et non périmées, puis suivant la dimension sociale pour ne conserver que celles appartenant à un cercle de connaissances de l'utilisateur, suivant la dimension de l'activité pour éliminer les informations qui n'ont pas de rapport avec ce que cherche l'utilisateur et enfin le filtre technique pour offrir des informations adaptées à la machine dont est issue la requête.

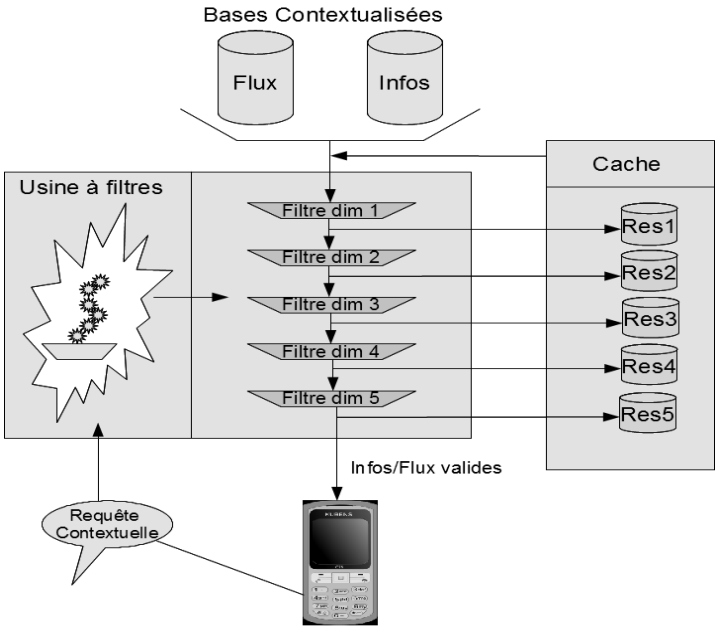

Figure 6. Système de filtrage contextuel en cascade. L'usine à filtres construit les requêtes SQL nécessaires au filtrage des données et les exécute dans l'ordre optimisé déterminé par les deux facteurs d'optimisation. Chaque étape filtrée est mise en cache pour réutilisation dans le cas de requêtes ultérieures proches

Chacune de ces étapes de filtrage peut être mise en cache afin d'accélérer les futures requêtes similaires, comme présenté dans le schéma de la figure 6. Comme nous l'avons dit précédemment, l'utilisation du cache est soumise à un calcul de

distance entre les dimensions des contextes des requêtes. Ici, un cache ne peut être réutilisé que dans le cas où la cascade de requêtes ayant abouti à ce cache est suffisamment proche de la nouvelle requête. Dans le cas où la distance sur la dimension du premier filtre est trop importante, la requête est à exécuter à nouveau en entier; il est donc nécessaire de prendre en compte la fréquence de modification des filtres pour optimiser leur ordonnancement. L'ordre des filtres est déterminé par deux facteurs :

– un facteur d'élagage : le filtre permettant d'élaguer le plus de données doit être appliqué le plus tôt possible dans la liste, pour permettre de diminuer rapidement le volume des données à filtrer ;

– un facteur de réutilisation : le filtre susceptible d'être le plus fréquemment modifié doit être appliqué le plus tard possible dans la liste pour éviter d'avoir à exécuter à nouveau toute la cascade de filtres lors des requêtes ultérieures.

Ces deux règles d'ordonnancement des filtres permettent d'obtenir un temps optimum de résolution des requêtes contextuelles proches, mais forcent également à changer dynamiquement l'ordre des filtres de la cascade.

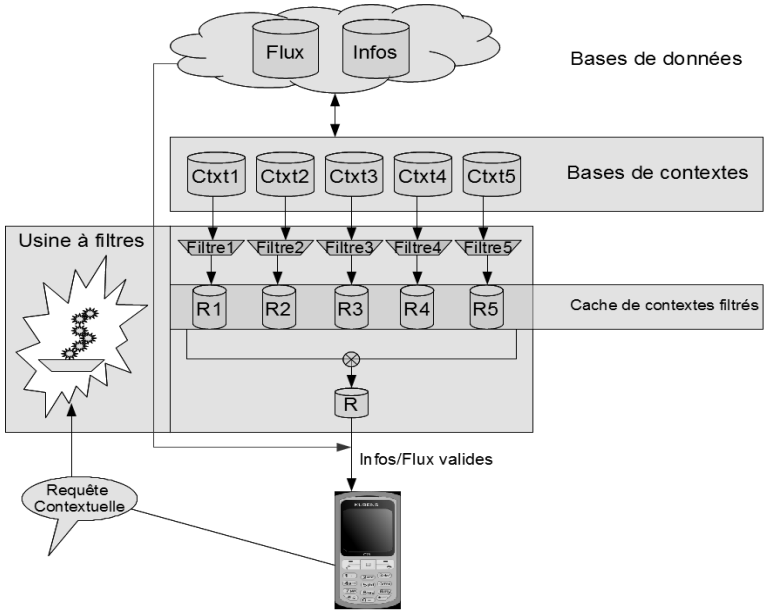

Figure 7. Système de filtrage contextuel en parallèle. L'usine à filtres construit les requêtes SQL nécessaires au filtrage des données et les exécute en parallèle. Chaque étape filtrée est mise en cache pour réutilisation dans le cas de requêtes ultérieures proches. Le croisement des cinq dimensions filtrées permet d'obtenir les informations et flux correspondant à la requête d'origine

5.3. Méthode de filtrage en parallèle

La seconde méthode considérée a pour but d'éviter la séquentialité des requêtes, afin d'éviter les problèmes de réutilisation de cache liés à la cascade. Cette modification de l'exécution des filtres entraîne des répercussions sur la structure de la base de données afin de bien séparer chaque dimension du contexte, rendant ainsi possible la distribution et parallélisation du filtrage de chacune de ces dimensions comme présenté à la figure 7. Le temps de filtrage contextuel n'est dans ce cas plus la somme des cinq requêtes séquentielles, mais bien uniquement le temps d'exécution de la plus lente.

De plus, l'utilisation des caches est simplifiée du fait que les requêtes ne dépendent plus les unes des autres contrairement au cas précédent, mais deviennent toutes autonomes. L'apport du cache dans ce cas est plus discutable, l'unique situation dans laquelle son apport est incontestable étant celle où aucune dimension contextuelle ne varie, et où les cinq dimensions filtrées sont donc tirées du cache. Le contrecoup lié à ces modifications de structure est l'apparition d'une jointure coûteuse entre les différents résultats intermédiaires de filtrage pour parvenir au résultat final.

6. Tests préliminaires

6.1. Première expérimentation

6.1.1. Objectif

Pour nous permettre de valider d'un point de vue technique nos idées concernant le filtrage contextuel d'informations, nous avons mis en place un dispositif de tests et mené quelques expérimentations. Le but premier était de pouvoir comparer l'efficacité des différents modes de requêtes (brute, en cascade et en parallèle) avec et sans utilisation de cache, afin de confirmer notre intuition que l'utilisation d'un cache pour les requêtes nécessitant des calculs de distance permet une amélioration des performances du système.

6.1.2. Contexte logiciel

La plate-forme mise en place se compose d'une base de données sous MySQL 5.1.41, et d'un client développé en java permettant la simulation de l'application client mobile d'une part, et la simulation du serveur distant (gérant la base de données et le cache) d'autre part.

6.1.3. Jeu de test

Notre jeu de test se compose de 64 344 informations tirées de deux bases de données internes de dépêches de journaux français, réparties en 124 flux d'informations. Chacun de ces flux et chacune des informations ont été contextualisés : nous les avons géolocalisés (à l'aide du service YahooPlaceFinder), temporalisés, leur avons ajouté des mots-clés, des restrictions techniques et des

informations sociales (information libre, gratuite, etc.). Les tables de la base de données ont été indexées suivant plusieurs dimensions, avec la méthode d'indexation d'arbre binaire.

6.1.4. Déroulement

Les tests se sont déroulés de la manière suivante. Des séries de requêtes ont été émises du client vers le serveur, suivant les différents modes à tester (interrogation directe de la base de données, en cascade ou en parallèle).

– requêtes identiques ou non en mode direct, sans cache (1) ;

– requêtes identiques en mode direct avec cache (2) ;

– requêtes identiques ou non en mode cascade sans cache (3) ;

– requêtes identiques en mode cascade avec cache (4) ;

– requêtes (par dimension) dont la cinquième dimension contextuelle varie en mode cascade avec cache (5) ;

– requêtes identiques ou non en mode parallèle sans cache (6) ;

– requêtes identiques en mode parallèle avec cache (7) ;

– requêtes (par dimension) dont une dimension contextuelle varie en mode parallèle avec cache (8).

Pour les tests effectués en faisant varier chaque dimension de contexte, seules les dimensions pertinentes sont montrées dans les résultats, celles qui ne sont pas indiquées se comportent comme la dimension « activité ».

Ce jeu de requêtes va pouvoir nous permettre de comparer les temps de réponse entre une requête SQL brute (1), cette même requête décomposée pour l'utilisation de cache en cascade (3) et enfin pour l'utilisation de cache en parallèle (6), mais sans utiliser les caches dans un premier temps. Puis dans un second temps, en utilisant les caches complets (4, 7) pour pouvoir comparer les gains et pertes de performance. Des résultats dégradés sont attendus dans les cas de la décomposition des requêtes sans utilisation des caches, tandis qu'un gain en performance est prévu pour ces mêmes requêtes avec utilisation des caches. Enfin et afin de juger les différences entre les deux modes de caches, des requêtes construites en faisant varier une dimension du cache sont exécutées (5, 8a, 8b).

6.1.5. Résultats

Les résultats obtenus (figure 8) sont dans l'ensemble assez proches de nos attentes, à savoir que les modes avec cache en cascade et parallèle sont plus performants dans le cas de requêtes proches, permettant la réutilisation partielle ou totale des caches. Cependant, nous pouvons constater les résultats suivants nécessitant discussion et que nous traitons dans la partie suivante :

– notre utilisation du cache dans le cas de la requête directe semble n'apporter aucune amélioration ;

– les cas cascade sans cache et direct sans cache sont très proches, et plus efficaces que le parallèle sans cache ;

– le cas parallèle avec réutilisation de quatre caches sur cinq n'est pas aussi efficace suivant la dimension à recalculer ;

– l'ordre d'application des filtres dans le cas de la cascade a une influence non négligeable (les résultats présentés sont ceux effectués dans l'ordre optimal).

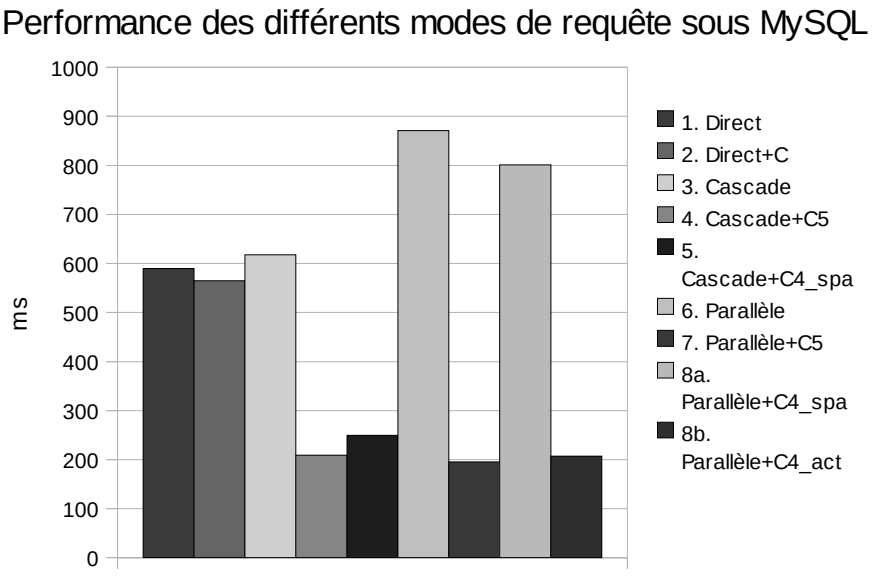

Figure 8. Diagramme des résultats de l'exécution moyenne des requêtes suivant les différents modes de filtrage, avec ou sans cache, et en introduisant des changements dans le contexte ou non

6.1.6. Discussion

Les quatre résultats cités précédemment s'expliquent assez facilement mais soulèvent des questions pour la suite de notre étude. Tout d'abord, le premier cas relève sans aucun doute du problème des vues non matérialisées dans MySQL, qui a un impact sur tous nos résultats mais se ressent plus particulièrement sur celui-ci. Il existe des méthodes permettant de contourner le problème, également présent sous PostgreSQL, en combinant des méthodes pour gérer la création de tables de cache permanentes alimentées périodiquement par les vues. Cependant cette méthode peut elle aussi s'avérer coûteuse en temps, ce que nous souhaitons éviter le plus possible.

Le second résultat s'explique facilement également. Les requêtes en cascade, dans le cas de la non-utilisation des caches aboutissent au final à la même requête

que la méthode directe, alors que le filtrage en parallèle se termine par une jointure complexe des différentes tables.

La différence de performance constatée en 3 doit être rattachée au cache interne de MySQL. Certaines des vues étant de simples sélections, elles peuvent être mises en cache par le SGBD directement, tandis que d'autres filtres de dimensions (spatial particulièrement) comportant des appels de fonction et calculs de distance ne peuvent être gérés par le cache interne du SGBD. Ce problème est également lié au problème de matérialisation des vues évoqué précédemment, et demande à être testé avec des vues matérialisées.

Enfin, le dernier point soulevé dans les résultats confirme notre intuition concernant l'importance de l'ordre des filtres en cascade. Cet ordre a des répercussions directes sur l'ordre des jointures effectuées, et affecte donc particulièrement les performances. Un filtre permettant d'élaguer l'ensemble des solutions de façon maximale sera à effectuer le plus tôt possible dans la cascade.

Nous avons pu constater que le filtrage spatial consomme énormément de temps, et élague fortement l'espace des résultats. Il nous semble donc intéressant de nous pencher sur le cas de SGBD optimisés pour les Systèmes d'informations Géographiques tels que PostGIS, pour améliorer les performances de filtrage spatial et obtenir des résultats plus satisfaisants.

De manière générale, l'utilisation des caches apporte également son lot de complications, à commencer par la durée de validité d'un cache et la cohérence entre les données réelles et en cache.

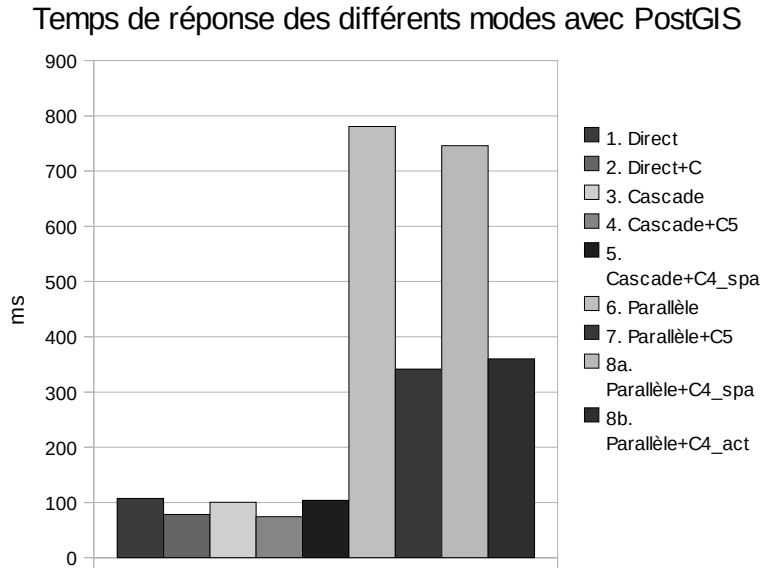

Figure 9. Diagramme des résultats de l'exécution moyenne des requêtes contextuelles suivant les différents modes de filtrage, avec PostGIS

6.2. Utilisation de PostGIS

Nous avons reconduit dans un second temps la même expérimentation, mais basée sur le SGBD PostGIS, afin de bénéficier de ses avantages pour l'indexation spatiale. Nous présentons figure 9 les courbes des résultats de ce second test.

Ces résultats sont semblables à ceux précédemment obtenus, avec un gain de performance important imputable à l'optimisation géographique. On constate cependant que le mode parallèle bénéficie peu de ces gains, le coût de la jointure finale étant beaucoup plus important que sous MySQL.

Le mode de cache en cascade permet a priori un gain de temps plus important, mais dans un plus petit nombre de situations que le cache en parallèle.

On observe également que l'utilisation d'un tel cache est alors peu intéressante par rapport à une requête directe sur la base de données.

7. Conclusion

Nous avons présenté dans cet article nos propositions de systèmes de filtrage contextuel avec utilisation de caches afin d'améliorer les temps de réponse aux requêtes contextuelles des utilisateurs de la plate-forme Vietaville. Ces deux systèmes de filtrage, par cascade ou en parallèle, couplés à un système de cache, permettent la réutilisation partielle des résultats des recherches précédentes.

Les principaux problèmes constatés viennent des limites du SGBD utilisé et des optimisations sont encore possibles, notamment par l'introduction de mécanismes de vues matérialisées permettant d'éviter le calcul des vues à chaque accès.

La gestion des caches de requêtes est actuellement effectuée par un module java, et ne semble pas consommer excessivement de temps (de l'ordre de 10 ms par recherche de cache). Cependant, notre cache était relativement limité en taille, et lors d'un passage à l'échelle, il nous semble important de souligner que l'optimisation de la gestion des caches risque de soulever de nouveaux problèmes, tels que la durée de vie des caches. Il en va de même pour la maintenance des caches, qui même si elle peut se dérouler en parallèle de leur utilisation, nécessitera de se pencher sur les questions d'optimisation des mises à jour de ces derniers pour garantir le maintien de la validité et de la cohérence des données.

Bibliographie

Ajanki A., Billinghurst M., Jarvenpaa T., Kandemir M., Kaski S., Koskela M., Kurimo M., Laaksonen J., Puolamaki K., Ruokolainen T., Tossavainen T. (2010). Ubiquitous Contextual Information Access with Proactive Retrieval and Augmentation, *Proceedings of MLSP 2010, IEEE International Workshop on Machine Learning for Signal Processing*, p. 95-100.

Arens Y., Knoblock C.A. (1994). Intelligent caching: selecting, representing, and reusing data in an information server, *Proceedings of the third international conference on Information and knowledge management* (CIKM'94), ACM, New York, USA, 433-438.

Baldauf M., Dustdar S., Rosenberg F. 2007. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2, n° 4.

Bolchini C., Curino C.A., Quintarelli E., Schreiber F.A., Tanca L. (2007). A data-oriented survey of context models. *SIGMOD Rec.* 36, 4, p. 19-26.

Boughanem M., Savoy J. (2008). *Recherche d'information : état des lieux et perspectives*, Hermès Science Publications.

Bradley N.A., Dunlop M.D. (2005). Toward a multidisciplinary model of context to support context-aware computing, *Human-Computer Interaction* 20, 4, p. 403-446.

Chaudhuri S., Krishnamurthy R., Potamianos S., Shim K. (1995). Optimizing queries with materialized views. *11th International Conference on Data Engineering (ICDE'95)*, p.190.

Dey A.K. (2001). Supporting the construction of context-aware applications. *Dagstuhl Seminar on Ubiquitous Computing*, Dagsthul, Germany.

Dey A.K., Abowd G.D. (1999). *Towards a better understanding of context and context-awareness.* Technical ReportGIT-GVU-99-22, Georgia Institute of Technology.

Funk H.B., Miller CA. (1997). Context sensitive interface design. *Proceedings of Context '97*.

Gronau N., Laskowski F. (2003). Using case-based reasoning to improve information retrieval in knowledge management systems, *Advances in Web Intelligence*, p. 954-954.

Guha R.V., McCarthy J., (2003). Varieties of Contexts , *Fourth International Conference on Contexts*.

Ilarri S., Mena E., Illarramendi A. (2010). Location-dependent query processing: Where we are and where we are heading. *ACM Computing Surveys* 42, 3.

Kaenampornpan M., O'Neill E. (2004). Modelling context: an activity theory approach, *Ambient Intelligence*, 367-374.

Kirsch-Pinheiro M., Villanova-Oliver M., Gensel J., Martin H. (2005). Context-aware filtering for collaborative web systems: adapting the awareness information to the user's context, *Proceedings of the 2005 ACM Symposium on Applied Computing* ACM, New York, 1668-1673.

Korpipää P., Häkkilä J., Kela J., Ronkainen S., Känsälä I. (2004). Utilising context ontology in mobile device application personalisation, *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, p.133-140, College Park, Maryland.

Kumar F., Gopalan S., Sridhar V. (2005). Context enabled Multi-CBR based Recommendation Engine for E-commerce, *Proceedings of the IEEE International Conference on e-Business Engineering*, p. 237-244.

Lemlouma T., Layaïda N. (2004). Context-Aware Adaptation for Mobile Devices, *IEEE International Conference on Mobile Data Management*, Berkeley, California, USA.

Martins D.S., Biajiz M., do Prado A. F., de Souza W.L. (2009). Implicit relevance feedback for context-aware information retrieval in UbiLearning environments, *Proceedings of the 2009 ACM symposium on Applied Computing*, 659-663.

Miranker D., Taylor M.C., Padmanaban A. (2002). A tractable query cache by approximation, *Lecture Notes in Computer Science*, vol. 2371/2002, p.178-187.

Ren Q., Dunham M.H., Kumar V. (2003). Semantic Caching and Query Processing, *IEEE Transactions* on *Knowledge* and *Data Engineering*, 192-210.

Ren Q., Dunham M.H. (2000). Using semantic caching to manage location dependent data in mobile computing, *Proceedings of the 6th annual international conference on Mobile computing and networking* (MobiCom '00). ACM, New York, USA, 210-221.

Tamine-Lechani L., Boughanem M., Daoud M. (2009). Evaluation of contextual information retrieval effectiveness: overview of issues and research, *Knowledge and Information Systems* 24, n° 1, 7, p. 1-34.

Wigelius H., Väätäjä H. (2009). Dimensions of Context Affecting User Experience in Mobile Work, *Proceedings of the 12th IFIP TC 13 international Conference on Human-Computer interaction: Part II* (Uppsala, Sweden, August 24 - 28, T. Gross, J. Gulliksen, P. Kotzé, L. Oestreicher, P. Palanque, R. O. Prates, and M. Winckler, Eds. Lecture Notes In Computer Science. Springer-Verlag, Berlin, Heidelberg, p. 604-617.

Weiβenberg N., Voisard A., Gartmann R. (2004). Using ontologies in personalized mobile applications, *Proceedings of the 12th Annual ACM international Workshop on Geographic information Systems* (Washington DC, USA, November 12-13, GIS'04. ACM, New York, p. 2-11.

White S., Feiner S. (2009). SiteLens: situated visualization techniques for urban site visits, *Proceedings of the 27th international conference on Human factors in computing systems*.

Xu J., Tang X., Lee D.L., Hu Q. (1999). Cache Coherency in Location-Dependent Information Services for Mobile Environment, *Proceedings of the First International Conference on Mobile Data Access* (MDA'99), Hong Va Leong, Wang-Chien Lee, Bo Li, and Li Yin (Eds.). Springer-Verlag, London, UK, p. 182-193.

Zheng B., Xu J., Lee D.L. (October 2002). Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments, *IEEE Transactions on Computers* 51, 10, p. 1141-1153.

Zheng B. Lee D.L. (2001). Semantic Caching in Location-Dependent Query Processing, *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases* (SSTD'01), Christian S. Jensen, Markus Schneider, Bernhard Seeger, and Vassilis J. Tsotras (Eds.). Springer-Verlag, London, UK, p. 97-116.

Zetie C., (2002) *Context: Delivering the true value of enterprise mobile computing.* http://www.unwiredexpress.com/products/downloads/ContextWPv2.pdf