Enterprise context-awareness : empowering service users and developers

Bachir Chihani

THESE DE DOCTORAT CONJOINT TELECOM SUDPARIS et L'UNIVERSITE PIERRE ET MARIE CURIE

Spécialité : Informatique et Télécommunication

Ecole doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

Bachir Chihani

Pour obtenir le grade de
DOCTEUR DE TELECOM SUDPARIS

La contextualisation en entreprise : mettre en avant utilisateurs et développeurs

Soutenue le 05/12/2013
devant le jury composé de :

Prof. Dr. Noël Crespi	**Professeur à Telecom SudParis**	Directeur de thèse
Dr. Emmanuel Bertin	**Expert senior à Orange Labs P&S**	Encadrant
Prof. Dr. Yacine Ghamri	**Professeur à l'université de La Rochelle**	Rapporteur
Prof. Dr. Jiangtao Wen	**Professeur à l'université de Tsinghua**	Rapporteur
Prof. Dr. Sebastien Tixeuil	**Professeur à Paris 6**	Examinateur
Dr. Roberto Minerva	**Manager à Telecom Italia**	Examinateur
Dr. François Toutain	**Docteur à Orange Labs P&S**	Examinateur
Dr. Ahmed Bouabdallah	**Enseignant-chercheur à Telecom Bretagne**	Examinateur

Thèse n° 2013TELE0029

Enterprise context-awareness: empowering service users and developers

Bachir Chihani

Department of Mobile Multimedia Network and Services

Institut Mines-Telecom, Telecom SudParis

A thesis submitted for the degree of

Doctor of Philosophy

I would like to dedicate this thesis to my loving parents ...

Acknowledgements

My Ph.D thesis has been jointly carried out at BIZZ Department as part of the ViVA team, Orange Labs P&S and the Wireless Networks and Multimedia Services Department at Telecom SudParis from 2010. I indulged in my passion for research at these two institutions while refining my technical prowess. My Ph.D journey created many opportunities to work with many people at different locations. I interacted with many individuals who encouraged me in many ways, such as through discussion on ideas and new concepts, and brainstorming. This dissertation resulted from collaborative work, and I want to acknowledge them here. I would like to thank my supervisors, Emmanuel Bertin, and Noel Crespi for their guidance and support during the research. We have enjoyed many vehement discussions in research problems, solutions and results. These two have given me an incredible perspective on numerous topics. For that and more, I am and will always be grateful. All colleagues at ViVA and NCIS have had a hand in guiding and helping me throughout my stay, my deepest heartfelt thanks to all of them. I am also thankful for the members of the service architecture group who inspired and encouraged me in many ways. We shared and discussed many topics in each team meeting.

Abstract

Context-aware applications must manage a continuous stream of contextual events according to dedicated business logic. Recent researches have focused on the proposal of several frameworks and platforms. However, the platform or framework behavior toward applications remains largely predefined. This thesis attempts to address this challenge by extending the background works through the proposal of new concepts serving as a foundation for a flexible approach for building context-aware applications.

The thesis studies the use of contextual information in common applications and examines the state of the art in the research area of context-aware computing in terms of techniques for context modeling and reasoning. It concludes that existing approaches tend to tightly couple the semantic of context to the modeling approach and proposes a context-centric modeling approach allowing the creation of a graph-based representation where entities are connected to each other through links representing context. Unlike existing approaches, the context graph decouples the presentation and the semantics of context, leaving each application to manage the appropriate semantic for their context data. In addition, this modeling approach benefits from the graphic representation as a way to natively support social networking applications.

The concepts proposed by the thesis provide a foundation for an effective design of context-aware applications where the context-related operations can be easily separated from the business logic of an application and as a result supporting the evolution of each aspect without alteration to the other one. The thesis adopts well-established design principles in software engineering and a well-defined functional decomposition to design a reference model for context management to implement these concepts into a comprehensive architecture supporting a seamless integration of context-awareness into applications.

Some case studies are conducted for the evaluation of the proposed system in terms of its support for the creation of applications enhanced with context-awareness. A simulation study is performed to

analyze the performance properties of the proposed system. The results of these studies prove the validity and the applicability of the system as well as the underlying concepts.

The result of this thesis is the introduction of a novel approach for supporting the creation of context-aware applications through a framework that decouples context management from an application's business logic. It supports the integration of context-awareness to existing applications without the need for a complete modification of their internal behavior. It empowers developers as well as users to participate in the creation process, thereby reducing potential usability problems.

Abstract

Les applications contextuelles doivent gérer selon une logique appropriée un flux continu d'informations de contexte. Les travaux de recherche récents du domaine se sont concentrés sur la fourniture de plateformes et de frameworks. Pourtant, le comportement de ces plateformes et frameworks reste largement prédéfini. Cette thèse adresse cette question en étendant les travaux existants par de nouveaux concepts pour fonder une approche flexible de création d'applications contextualisées.

Cette thèse examine l'état de l'art des travaux de recherche en contextualisation en termes de technique de modélisation et de traitement des informations de contexte. Elle étudie l'utilisation des informations de contexte dans des applications courantes et propose une nouvelle approche de modélisation de contexte permettant la création d'une représentation en graphe oú les entités sont connectées entre elles á travers des liens dérivés des informations de contexte. Contrairement aux approches existantes oú la sémantique du contexte est fortement liée á la méthode de modélisation, le graphe de contexte découple la présentation et la sémantique de contexte laissant chaque application gérer une sémantique de contexte qui lui est propre. De plus, cette approche bénéfice de la représentation en graphe pour nativement supporter les applications liées aux réseaux sociaux.

Les concepts proposés par la thèse permettent une conception simplifiée des applications contextuelles oú les opérations de gestion de contexte sont séparées de la logique métier de l'application, permettant ainsi de faire évoluer un aspect sans modifier l'autre.

La thèse adopte des principes répandus de conception logicielle et une décomposition fonctionnelle bien définie pour concevoir un modèle de référence implémentant ces concepts dans une architecture complète pour supporter l'intégration de comportement contextuel dans les applications.

Quelques études de cas sont menées pour l'évaluation du système proposé en terme de support pour la création d'applications dont le comportement est contextualisé. Le résultat de ces études montre la

validité et l'applicabilité du système tout comme des concepts sous-jacents.

Le résultat de la thèse est l'introduction de nouvelles approches de création d'applications contextuelles á travers un framework qui découple la gestion du contexte de la logique métier de l'application. L'approche proposée permet l'intégration de comportements contextuels aux applications existantes sans nécessiter une modification complète de leurs composants internes. Elle met ainsi en avant le développeur mais aussi l'utilisateur afin de réduire des problèmes potentiels d'utilisabilités.

Publications

International Journals

1. B. Chihani, E. Bertin, N. Crespi, "Programmable Context Awareness Framework," Journal of Systems and Software (JSS'13), ISSN 0164-1212, 2013, http://dx.doi.org/10.1016/j.jss.2013.07.046.
2. B. Chihani, E. Bertin, N. Crespi, "Android-based QoE management framework," WiP, IEEE Pervasive Computing vol. 11, no. 4, October-Decembrer 2012.
3. B. Chihani, E. Bertin, I. Salsabila Suprapto, J. Zimmermann, N. Crespi, "Enhancing existing communication services with context-awareness," Journal of Computer Networks and Communications, vol. 2012, Article ID 493261, 10 pages, 2012, doi:10.1155/2012/493261.
4. B. Chihani, E. Bertin, F. Jeanne, N. Crespi , "HEP: context-aware communication system," International Journal of New Computer Architectures and their Applications (IJNCAA), vol. 1, no. 1, June 2011.

International Conferences

5. B. Chihani, E. Bertin, D. Collange, N. Crespi, "User-centric Quality of Experience Measurement," Fifth International Conference on Mobile Computing, Applications and Services (MobiCASE'13), Paris, France, November 2013.
6. B. Chihani, E. Bertin, N. Crespi, "A User-Centric Context-Aware Mobile Assistant," 17th International Conference on Intelligence in Next Generation Networks (ICIN'13), Venice, Italy, October 2013.
7. B. Chihani, E. Bertin, N. Crespi, "A graph-based context modeling approach," 4th International Conference on SmArt COmmunications in NEtwork Technologies (SaCoNeT'13), Paris, France, June 17-19, 2013.
8. B. Chihani, E. Bertin, N. Crespi, "Decoupling Context Management and Application Logic: a new Framework," 2nd IEEE WoWMoM Workshop on

the Internet of Things: Smart Objects and Services (IoT-SoS'13), Madrid, Spain, June 2013.
9. B. Chihani, E. Bertin, N. Crespi, "Enhancing M2M communication with cloud-based context management," NGMAST 2012, Paris, France, September 2012.
10. B. Chihani, E. Bertin, N. Crespi, "A Comprehensive Framework for Context-Aware Communication Systems," International Conference on Intelligence in Next Generation Networks (ICIN'11), Berlin, Germany, October 2011.
11. B. Chihani, E. Bertin, F. Jeanne, N. Crespi, "Context-Aware Systems: A Case Study, Proceedings of International Conference on Digital Information and Communication Technology and its Applications (DICTAP'11), Dijon, France, June 2011.

Patents

- B. Chihani, K. Laghari, N. Crespi, QoE Based Framework for the assessment of Multimedia Services on Android Based Smart Phones. N INPI: FR1256177, 2012.
- B. Chihani, E. Bertin, Framework de gestion des informations contextuelles. N INPI: FR1254372, 2012.
- B. Chihani, E. Bertin, Improved provision of contextual information. Europe EP20120190063, 2011.

Contents

List of Figures

French summary

Dans les dernières années des changements significatifs ont transformé les équipements informatiques à la suite de l'émergence des téléphones intelligents équipés de plusieurs type de capteurs ainsi que la grande disponibilité de connectivité internet. Ces avancées ont permet la création de nouvelle forme d'interactions entre ordinateurs et utilisateurs qui vont au-delà des interactions traditionnelles basées sur des requêtes explicite de l'utilisateur. Ces nouvelles formes se basent sur la considération d'informations additionnelles désigné par information de contexte et décrivant la situation de l'utilisateur ou l'état de son environnement. L'utilisation de ces informations permet aux applications de mieux aligner leur comportement avec les besoins de l'utilisateur. Ce nouveau type d'applications sont désignées par applications contextuelles, ils doivent souvent effectuer un certain nombre d'opération liés à la gestion du contexte (ex. acquisition, présentation, stockage). A cause de la complexité de ces opérations, on ne peut développer des applications contextuel en appliquant des méthodes traditionnelles de développement logicielle telle qu'en cycle V où une description préalable du logiciel doit être formalisée, après un document de spécification est généré et seulement maintenant que l'implémentation de l'application peut commencer. De nouveaux outils doivent être fournis aux développeurs afin de les assister pendant la création d'applications contextuelles. En plus, il est important de faciliter l'insertion des utilisateurs dans le processus de création, par exemple en fournissant des mécanismes permettant la personnalisation du comportement contextuel de l'application. De plus, à cause de la sensibilité des informations de contexte utilisateurs, ces outils de support doivent fournir des mécanismes de contrler d'accès adéquate.

Motivation

Les applications contextuelles gérant un flux continue d'information d'évènement de contexte dans un environnement dynamique ce qui influence le processus de développement de ces applications. Les travaux de recherche se sont concentrés dans la facilitation du développement d'application à travers la proposition de plusieurs framework et plateformes. Ces outils aident à guider le processus de développement avec la réutilisation de composants dans le but de faire gagner du temps de développement et cacher des détails techniques inutiles via une couche d'abstraction. Cependant les développeurs ont généralement une vue limitée sur l'environnement d'exécution de leur application, ainsi ils peuvent facilement louper certain aspects de l'application au moment de développement qui peuvent être important du point de vue utilisateur.

En plus, les changements aux comportements de l'application ou dans la variété de son contexte peut ne pas être possible au moment de son exécution. Aussi la personnalisation du comportement adaptatif par l'utilisateur peut ne pas être suffisante dans certain cas et une intervention (ex. du développeur, administrateur) peut être nécessaire.

Les travaux en contextualisation doivent déplacer leur concentration pour fournir plus de transparence aux autres acteurs (ex. les utilisateurs) pour permettre la personnalisation des comportements contextualisés des applications. Aussi, d'éventuels prérequis liés à l'environnement d'exécution et concernant la modification de la structure de l'application doivent être prise en compte. Cela devra combler l'espace entre le cas où un comportement contextuel n'a pas été défini et le cas où la sélection de la bonne action est ambigu. Aussi, pouvoir développer des applications contextuels en mode centré utilisateur devra permettre d'éviter des problèmes d'utilisabilité.

Publications

Cette thèse a pour but de faciliter la création d'applications contextuelles et étendre le rayon d'acteurs impliqués dans cette activité pour inclure les utilisateurs finaux sur différents niveaux comme la configuration, personnalisation du comportement contextuel de ces applications. En plus, elle permet aux développeurs

d'être capables de modifier la configuration de l'application aisément sur toutes les phases du cycle de vie de celle-ci.

Cette section introduit brièvement les contributions en pointant vers les articles correspondants, le chapitre 4 présentera en détails la description des contributions.

Les deux articles 10 et 11 abordent les motivations pour étudier l'informatique contextuelle, examinent les importants travaux de recherche menés, conceptualisent et formalisent les problèmes de recherches. Aussi, ils identifient les challenges en face de ce type d'applications à travers l'examination de comment le contexte est utilisé dans les applications contextuelles et où le contrle adaptatif est réalisé. La contribution consiste dans la proposition d'une nouvelle approche pour classifier les caractéristiques des applications contextuelles, aussi sur la proposition de nouvelle vision et modèle de référence pour les applications contextuelles.

Les articles 3 et 4 abordent les problèmes de recherche liés aux services de communication contextuelles dans un environnement entreprise et proposent une conceptualisation de référence pour ce type particulier de service. Celle-ci est concrétisée en un framework et implémentée en une architecture générique afin de guider la création des applications contextuelles et sera utilisé pour implémenter quelques prototypes de services de communications améliorés avec un comportement contextuel.

Dans les articles 1 et 8, on propose la conception d'un framework composé fonctionnellement en plusieurs composants mappant certaines opérations de gestion de contexte. Ce framework supporte la réorganisation des composants de faon flexible permettant aux développeurs de combiner la création des procédures de traitement de contexte spécifique à leurs applications.

Dans l'article 9, on propose d'adapter cette architecture pour pouvoir gérer des informations de contexte (i.e. présentation, traitement) dans un environnement Cloud. La proposition permet de fournir aux développeurs une architecture flexible et évolutive pour supporter un gros volume de données. Aussi, de fournir un langage de description permettant la définition de procédure de traitement de contexte et d'adaptation.

L'article 7 propose de représenter le contexte comme donnée relationnels qui peut être utilisé pour connecter deux entités entre elle, ainsi introduit un nouvel concept du graphe de contexte qui se base sur cette représentation. Aussi,

l'article décrit la conception et développement d'un framework de gestion de graphe capable de gérer les informations de contexte et les méthodes d'accès basé sur exploration de graphe.

Une autre contribution consiste dans l'intégration de nouvelles fonctionnalités dans ce framework afin de permettre aux utilisateurs de définir des règles pour contrler l'exécution de services. L'article 6 aborde les problèmes d'utilisabilités liés aux applications mobiles qui résulte du manque de coopération entre ces applications. L'article introduit la conception et développement d'un framework de composition de service contextuel: composer des services pour construire des applications et contrler leur exécution à l'aide de condition à base de contexte.

Les articles 2 et 5 introduisent un outil de mesure de la qualité d'expérience qui permet de générer un score représentant une évaluation de l'expérience utilisateur d'un service multimédia. L'outil combine de multiples informations disponible sur le mobile utilisateur entre autre des informations de contexte afin de personnaliser le calcul du score.

En plus, certains des résultats ont été brevetés.

État de l'art

Le terme contexte est utilisé dans plusieurs domaines de recherche, notamment en linguistique. En informatique, le contexte représente toute information liée à une entité (ex. utilisateur, équipement) qui peut être capturé par une application dans le but d'adapter le comportement de celle-ci selon l'état de l'utilisateur. Des exemples d'information de contexte inclus:

- L'identité qui caractérise l'utilisateur avec un identifiant unique et explicite dans le domaine de l'application.
- La localisation qui correspond aux données géographique et relations spatiales (ex. données de positionnement, orientation, etc.).
- Les états qui se réfèrent aux facteurs physiques, signaux vitaux, fatigue, état émotionnel, etc.
- Des informations qui décrivent les relations entre les utilisateurs tels que l'amitié, ou relation entre utilisateur et objet ou environnement tel que la possession.

La figure suivante 1 illustre les différents types d'information de contexte sur l'utilisateur et son environnement et qui peuvent être utilisé par une application dans le cas de contextualisation.

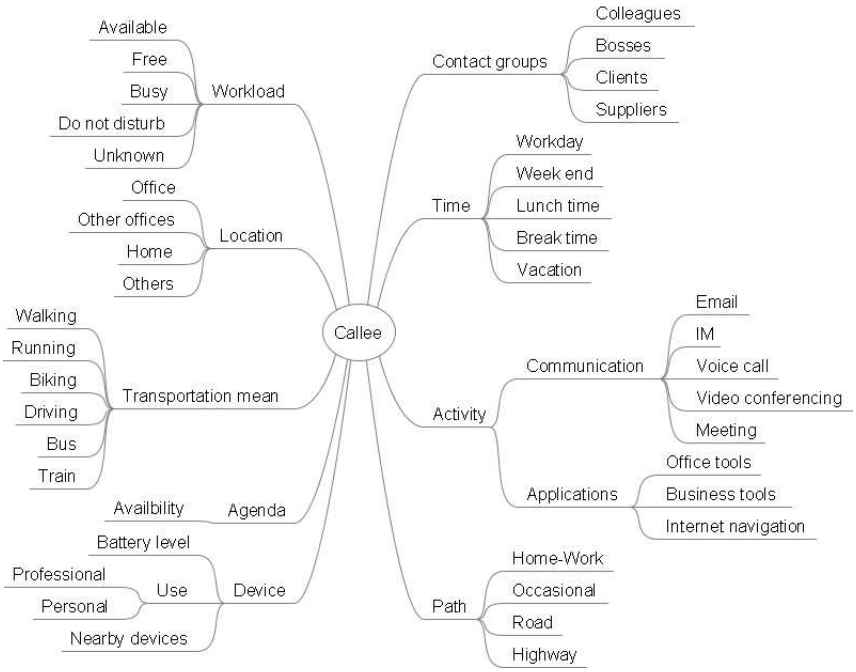

Figure 1: Exemple de données de contexte

Le contexte a été toujours considéré comme un élément ou attribut associé à une seule entité. Par exemple, la localisation GPS est une variable don't la valeur est attribué à l'entité localisée à cette position. La figure 2.b illustre le perspective centrée entité des informations de contexte caractérisée par des étoiles où chaque entité est représentée par le noeud centrale qui est entouré par des informations de contexte décrivant la situation de cette entité. Cependant, vu que l'information de contexte décrit souvent des relations entre entités alors c'est plus naturel de représenter cette information à l'aide d'un arc liant deux noeuds dont chacun représente une entité. Ainsi, un graphe est constitué offrant une perspective centré contexte (comme indiqué dans la figure 2.a) qui est plus global que la vue centrée entité où les entités sont considérés séparemment.

Les applications contextuelles sont des applications qui utilisent en plus des informations métiers d'autres types d'informations qui décrivent la situation de

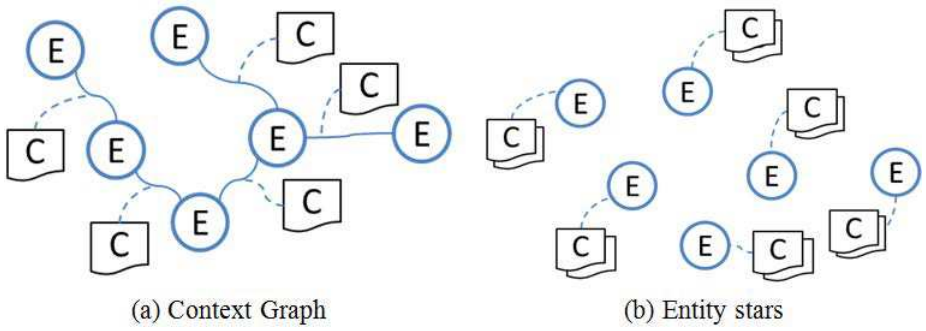

Figure 2: Context-centric vs. Entity-centric view

l'utilisateur avec qui l'application interagit. Elles doivent gérer selon une logique appropriée un flux continu de ces informations de contexte. Ces types d'applications sont traditionnellement classés par le type de métier qu'ils adressent. Le tableau suivant illustre cette classification.

Communication Socialight [25] a pour but de permettre à l'utilisateur de publier des messages sur une position géographique de faon à ce qu'ils soient visible à quiconque se trouvant pas loin de cette position. Calls.calm [26] permet aux utilisateurs de communiquer des informations sur leur situation et sur les canaux de communications sur lesquels ils sont disponibles pour tre contacter.

Assistant SECE [27] utilise divers informations de contexte pour permettre aux utilisateurs d'écrire des règles d'adaptation afin de gérer le comportement d'un service de communication. OnX [28] est un assistant mobile qui utilise les informations de contexte pour déclencher l'execution d'une action.

Tourisme utiliser l'historique des visités de l'utilisateur pour le guider vers de nouvelles ouvres artistiques [29] ou utiliser les informations sur les ouvres que l'utilisateur est en train de regarder pour lui présenter des informations pertinentes de faon conviviale [30].

Travail capturer des informations (ex. localisation, détails d'une panne) sur un incidant pour les fournir au bon moment au technicien, par exemple celui qui est le plus proche ou celui qui possède les bonnes compétences. Capturer et collecter des informations environmentales (ex. climat, bruit, force de signal) relative à une position géographique pour visualiser une

aggrégation de ces données sur carte [31].

Achats améliorer l'expérience utilisateur en fournissant des détails sur un produit, aider l'utilisateur à trouver son chemin pour récupérer un produit dans une supermarché [32].

Musée combiner des données de contexte pour assister les visiteurs en adaptant les informations fournies concernant un ouvre selon leurs intérts et connaissances [33].

La thèse propose une nouvelle classification par rapport à l'utilisation du contexte illustrée dans la figure 3. Cette proposition se base quant à elle par deux axes :

- Un axe horizontal correspondant à moment de l'adaptation au contexte est effectué par l'application, i.e. soit instantanément au moment de la disponibilité de l'information de contexte, ou de faon différé à la suite d'un traitement batch et transformation d'un ensemble d'information de contexte.
- Un axe vertical décrivant l'emplacement où est implémentée l'intelligence responsable de l'opération d'adaptation au contexte, i.e. soit l'intelligence est implémentée sur le client installé dans l'équipement de l'utilisateur, ou implémenté de faon centralisé sur la partie backend de l'application.

Le choix d'une option ou autre dans l'architecture de l'application contextuelle a une conséquence sur son comportement adaptatif. Par exemple, implémenté l'intelligence directement au niveau du client permet d'être plus interactif face aux changements de contexte de l'utilisateur mais rend la mise à jour de tout le parc d'applications installés une tche difficile. Aussi les équipements utilisateurs (surtout mobiles) disposent de peu de ressources ce qui limite considérablement la capacité de traitement. Dans l'autre partie, avoir une partie de l'intelligence de l'application sur un serveur permet d'avoir de considérable quantité de ressources et ainsi pouvoir implémenter plus de complexité de traitement. Cela dit, il faut prendre en compte la latence de la connexion réseau entre la partie client et celle serveur qui jouera considérablement sur la réactivité de l'application. Aussi, avoir une intelligence implémentée cté serveur limitera les possibilités de l'application dans le cas où l'équipement est utilisé en mode déconnecté.

Les travaux de recherche récents du domaine se sont concentrés sur la fourniture de plateformes et de frameworks permettant de faciliter la tche des développeurs

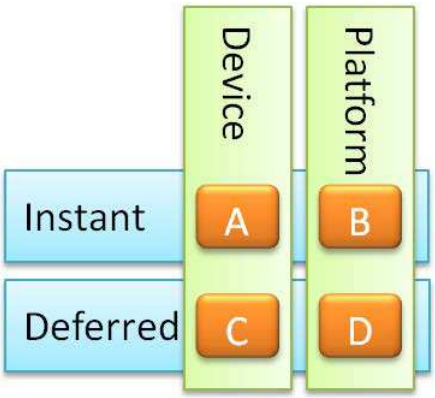

Figure 3: Classification des applications contextuelles selon deux axes

par l'abstraction des opérations de gestion de contexte. Les différents framework proposé par ces travaux sont soit spécifiques à un domaine d'applications particulier et ainsi ne peuvent être utilisé que pour la création de ce type d'application ou généralement applicable pour la création de différent type d'application. Un exemple du premier type de proposition est PLASH [71] qui est une plateforme facilitant le déploiement d'applications sensible aux contextes de l'utilisateur. La figure suivante 4 illustre l'architecture en couche de PLASH: la couche communication fournit une interface et un protocole aux applications tiers pour accéder aux ressources disponible sur la plateforme, elle supporte différent technologies de communications sans fil. La couche de données est responsable pour la représentation et stockage des informations de contexte, ainsi que la gestion des requêtes d'accès à ces informations. La couche service fournit un ensemble de services de base qui peuvent être utilisé par les applications déployées sur la plateforme, par exemple des mécanismes de contrle d'accès, services de gestion des bases de données, etc.

PLASH fournit un supporter intéressant aux développements d'applications sensibles particulièrement aux informations de localisation. Cependant, PLASH ne fournit pas de support à d'autres acteurs comme les utilisateurs pour participer à la configuration de l'adaptation au contexte de l'application.
Vimoware [73] un exemple de plateforme permettant la création d'applications de différent domaine. Techniquement, Vimoware est un middleware dont l'architecture est illustré dans la figure 5. Il permet le développement de services sur équipements

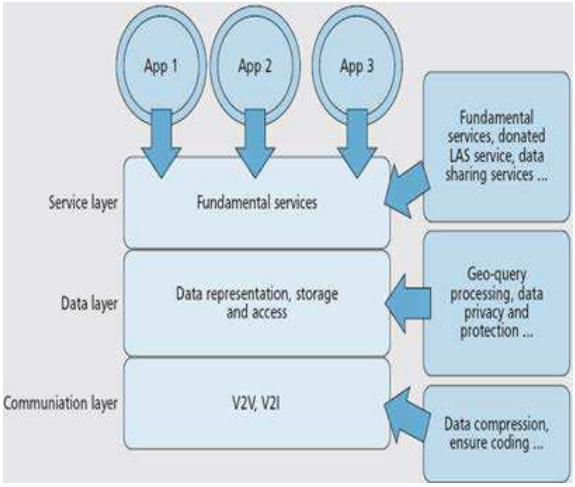

Figure 4: La platforme PLASH

mobiles et pouvoir les combiner et coordonner.

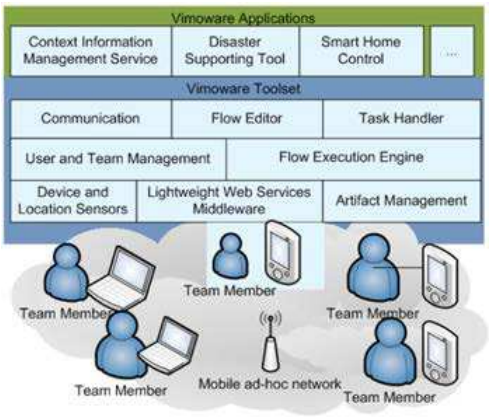

Figure 5: Les composants de l'architecture Vimoware

L'annonce et découverte des services se base sur une approche P2P de souscription et notification afin de permettre des notifications temps réel et limiter le trafic réseau engendré. La communication entre les différents composants se base sur le protocole HTTP et permet de distribuer des informations de contexte sur les équipements (ex. réseau) et utilisateurs (ex. compétences), gestion d'utilisateurs et de tches (ex. création, control), aussi fournit un service de communication (ex. im).

L'examination de l'état de l'art des travaux de recherche en contextualisation en termes de technique de modélisation et de traitement des informations de contexte, ainsi que l'étudie l'utilisation des informations de contexte dans des applications courantes permet de mieux comprendre les avantages et limites de ces travaux. En effet, le comportement de ces plateformes et frameworks reste largement prédéfini et ainsi limiter les possibilités qui peuvent être offert aux utilisateurs. Cette thèse adresse cette question en étendant les travaux existants par de nouveaux concepts pour fonder une approche flexible de création d'applications contextualisées.

Contributions

Cette thèse propose une nouvelle approche de modélisation de contexte permettant la création d'une représentation en graphe où les entités sont connectées entre elles à travers des liens dérivés des informations de contexte. Comme illustré dans la figure suivante 6 qui illustre une représentation d'un ensemble d'information de contexte liée à une communication entre deux personnes et incluant des informations sur les différent dispositifs utilisés.

Figure 6: Exemple d'une representation centrée contexte

Contrairement aux approches existantes où la sémantique du contexte est fortement liée à la méthode de modélisation, le graphe de contexte découple la présentation et la sémantique de contexte laissant chaque application gérer une

sémantique de contexte qui lui est propre. De plus, cette approche bénéfice de la représentation en graphe pour nativement supporter les applications liées aux réseaux sociaux par exemple pour représenter le graphe liée aux informations de contexte d'employé ainsi que les relations entre eux.

La procédure pour récupérer les informations de contexte d'un utilisateur se base sur l'api de parcours de graphe offerte par la plateforme de gestion de contexte. La figure suivante 7 illustre comment la plateforme gère ce type de requêtes:

- Vérifie si le noeud de démarrage est disponible dans la base de données, sinon les informations du noeud sont demandés auprès du fournisseur de données correspondant et le noeud est créé avec les informations reues, ainsi que les arcs sortants.
- Après avoir assuré de la disponibilité des informations du noeud, on vérifie quels sont les arcs qui satisfient les conditions exprimées dans la requête ainsi que leur validité.
- Tout arc est valide sera ajouté au parcour, dans le cas où un arc n'est pas valide ses informations sont redemandé de la source correspondantes.
- Après avoir exploré les arcs constituant le résultat d'analyse du noeud courant, c'est au tour des noeuds successives d'être explorés jusqu'à avoir tout parcouru.
- Les données constituées de ce parcours sont formattées en XML pour être envoyé comme résultat à l'application qui a initialement envoyé la requête.

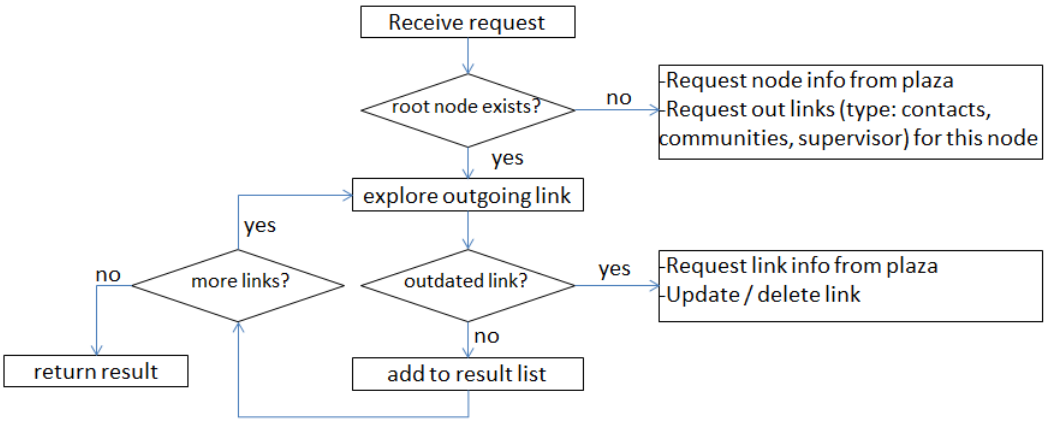

Figure 7: Exemple de gestion d'une requête de parcours de graphe

Les concepts proposés par la thèse permettent une conception simplifiée des applications contextuelles. La figure suivante 8 illustre le concept d'abstract-aggregate qui permet de concevoir une procédure de traitement de contexte qui sont complexe en construisant un réseau d'opérations de basé. L'ensemble de ces opérations est principalement : produce, filter, absract, select, aggregate et consume. La fonction produce permet de produire des informations de contexte de base, elle peut être implémentée par un fournisseur de contexte qui enveloppe un capteur par exemple afin de pouvoir s'intégrer avec l'API du framework. La fonction filter permet quant à elle d'éliminer ou réduire le bruit dans l'information de contexte récupérée par la fonction d'au-dessous. La fonction abstract permet de transformer des informations de bas niveau en informations abstraites plus expressives. La fonction select permet de sélection la meilleure information satisfaisant certains critère parmi un plusieurs informations de même type fournies par différents sources. La fonction aggregate permet d'agréger un ensemble d'information de contexte pour générer une composition qui sera par la suite exposée aux applications contextuelles. La dernière fonction de la série est consume, elle est censée être implémentée par les applications contextuelles qui représentent les consommateurs de contexte et pouvoir ainsi s'intégrer au framework.

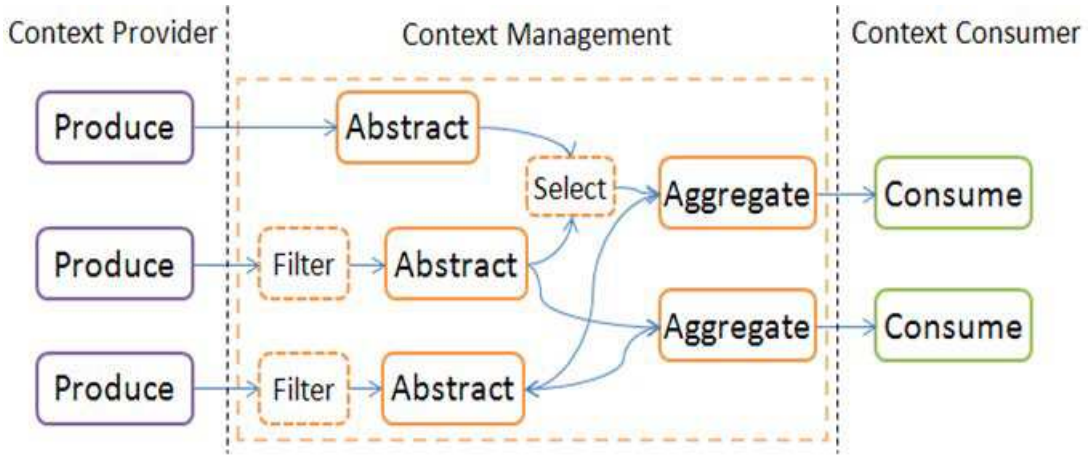

Figure 8: Illustration du réseau de traitement d'information de contexte

Des principes répandus de conception logicielle et une décomposition fonctionnelle bien définie sont adoptés pour concevoir un modèle de référence implémentant

ces concepts dans une architecture complète pour supporter l'intégration de comportement contextuel dans les applications. En plus, les opérations de gestion de contexte sont séparées de la logique métier de l'application, permettant ainsi de faire évoluer un aspect sans modifier l'autre. Cela grce à l'utilisation d'un langage spécifique à base de XML permettant la définition des opérations de gestion de contexte. L'instanciation d'un document conforme à cette définition permet de programmer au niveau du framework des opérations sur des données de contexte et coordonner ces opérations afin de réaliser des traitements complexes.

Listing : Le DTD de spcification XML de Abstract-Aggregate

```
<!DOCTYPE Definition [
<!ELEMENT Definition (Filter | Automata | Select | Rule)*>
<!ELEMENT Filter (Input, Output)>
<!ATTLIST Filter Type CDATA #REQUIRED >
<!ELEMENT Automata (State*, Start-State, End-State)>
<!ATTLIST Automata Id CDATA #REQUIRED Name CDATA #REQUIRED>
<!ELEMENT State (Transition)*>
<!ATTLIST State Name CDATA #REQUIRED >
<!ELEMENT Start-State (Transition)*>
<!ATTLIST State-State Name CDATA #REQUIRED >
<!ELEMENT End-State EMPTY>
<!ATTLIST End-State Name CDATA #REQUIRED >
<!ELEMENT Transition (Condition, Event*) >
<!ATTLIST Transition Dest CDATA #REQUIRED >
<!ELEMENT Select (Input, Output) >
<!ATTLIST Select Param CDATA #REQUIRED Opt CDATA #REQUIRED >
<!ELEMENT Input (Channel+) >
<!ELEMENT Output (Channel) >
<!ELEMENT Rule (Condition+, Event)>
<!ATTLIST Rule Id CDATA #REQUIRED Name CDATA #REQUIRED >
<!ELEMENT Condition ((UOP?, SimplCond),(BOP, UOP?, SimplCond)*) >
<!ELEMENT UOP EMPTY (NOT) "NOT">
<!ATTLIST UOP Type CDATA #REQUIRED >
<!ELEMENT BOP EMPTY >
<!ATTLIST BOP Type (AND | OR) "AND">
<!ELEMENT SimplCond (Channel) >
<!ATTLIST SimplCond Operator (EQ|NEQ|GE|LES) "EQ" Value CDATA #REQUIRED >
<!ELEMENT Event (Channel) >
<!ATTLIST Event Type(internal|external)"internal" Message CDATA #IMPLIED >
<!ELEMENT Channel EMPTY >
```

```
<!ATTLIST Channel prefix CDATA #REQUIRED suffix CDATA #REQUIRED >
]>
```

Quelques études de cas sont menées pour l'évaluation du système proposé en termes de support pour la création d'applications dont le comportement est contextualisé. Le résultat de ces études montre la validité et l'applicabilité du système tout comme des concepts sous-jacents.

HEP

HEP est un système de gestion de communication qui a pour but de superviser la disponibilité des utilisateurs afin de pouvoir contrler les communications entrantes. Il permet aussi de communiquer la charge de travail des utilisateurs à destination de leur contacts afin d'éviter des appels inutiles qui peuvent interrompre cet utilisateur. Le figure suivante 9 illustre l'architecture de HEP, pour l'utilisateur final, le système se présente sous forme d'un plugin qui peut se greffer sur une suite de communication (ici Outlook).

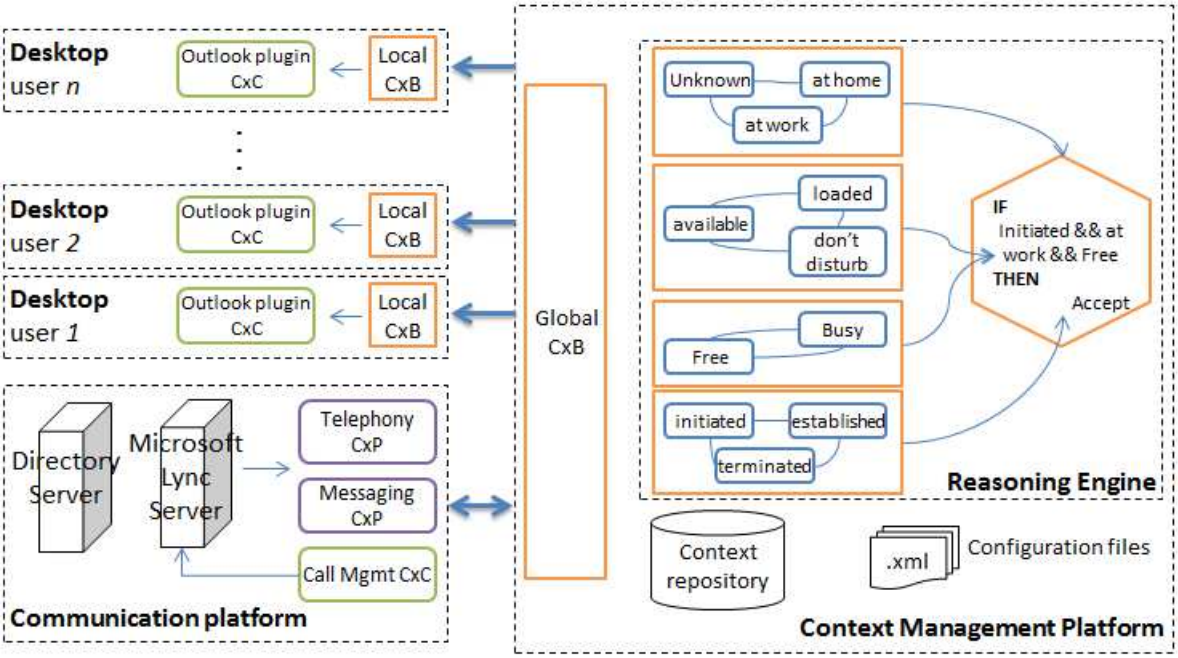

Figure 9: Un overview de l'rchitecture de HEP

Ce greffon permet d'étendre l'interface de base de la solution de communication avec des informations supplémentaires sur la disponibilité des contacts. Les différents éléments composant l'architecture sont:

- Un client PC responsable de récupérer des données depuis les différents services de communication disponibles à l'utilisateur (ex. email, im), il est aussi responsable du calcul du statut de disponibilité et de permettre à l'utilisateur de visualiser les statuts de ses contacts.
- La plateforme de gestion de contexte fournit des fonctions de stockage et distribution des informations de contexte, elle permet aux clients PC de souscrire et publier les statuts de leur utilisateur, et permet aux greffons Outlook de récupérer ces statuts.
- Une interface d'administration permet de gérer les règles générales de calcul des statuts.

ESD

ESD est une application de réseaux sociaux permettant de fournir une dimension social à l'annuaire de l'entreprise en étendant les liens statiques entre employés (ex. relations hiérarchiques) avec des liens maintenus dynamiquement en se basant sur l'activité des utilisateurs. Les informations exploitées sont fournies par:

- Un client LDAP concernant les informations sur profiles des employés et relations managériales,
- Un client Microsoft Lync pour fournir des informations liées aux communications.
- En plus, un client connecté à Plaza (un réseau social entreprise interne).

Les informations fournies sont utilisées pour maintenir un graphe de contexte exporté aux applications contextuelles.

Les différents composants de l'architecture d'ESD sont illustrés dans la figure suivante 10. La partie front-end de l'application est responsable de l'interaction avec les utilisateurs finaux via une interface web, aussi pour exposer les différentes fonctionnalités de l'application (ex. la recherche de profile par mots clés) aux utilisateurs. Sur la partie back-end sont déployés des sources d'informations qui sont en charge de collecter des informations depuis les serveurs d'annuaire et réseau social entreprise. Entre les deux environnements est déployée la plateforme

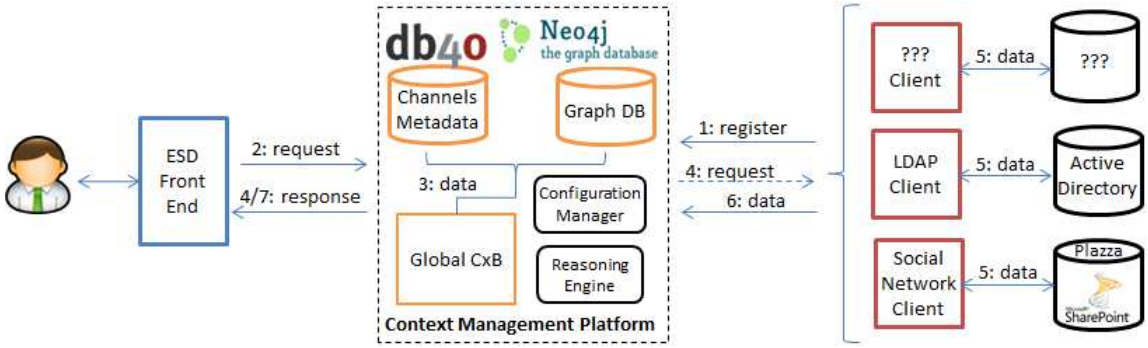

Figure 10: Un overview de l'architecture de ESD

de gestion de contexte afin d'intégrer les différentes informations collectées et fournir un service de stockage et gestion.

Conclusion

Le résultat de la thèse est l'introduction de nouvelles approches de création d'applications contextuelles à travers un framework qui découple la gestion du contexte de la logique métier de l'application. L'approche proposée permet l'intégration de comportements contextuels aux applications existantes sans nécessiter une modification complète de leurs composants internes. Elle met ainsi en avant le développeur mais aussi l'utilisateur afin de réduire des problèmes potentiels d'utilisabilités.

Chapter 1

Introduction

Over the last decade, phenomenal technology changes have transformed the personal computing devices due to the emergences of smartphones, the ubiquitous Internet connectivity, as well as the integration of a variety of sensors into these devices. These advances have enabled the creation of new forms of interactions between computers and users that go beyond the usual interactions based on explicit user requests. These new forms rely on information which describes the user situation or his/her environment, in order to fine tune the behavior of the underlying system, the way user requests are answered, or anticipating the user future needs. The new kind of applications are context-aware applications. They are functionally composed of a set of context-centric operations (e.g. context acquisition, modeling and processing) which are complex to build and maintain. These applications cannot simply be created with traditional software engineering patterns, for instance through the V-Model where the engineering process starts by a description of what needs to be done, then a specification is made out of that, then the application is implemented and tested, etc. Thus, new supportive frameworks and tools should be provided to developers in order to assist them during the creation of context-aware applications. In addition, it is important to enable the user's involvement in the creation of context-aware applications, for instance by providing mechanisms enabling the customization of the context-aware behavior. Furthermore, due to the sensitivity of the context information as they describe user information, adequate access control mechanisms should be provided by these support tools.

1.1 Motivation

Context-aware applications manage a continuous stream of contextual events in a highly dynamic environment that leverages the development process of these

applications. Research studies have focused on the empowerment of developers by providing early phase development support for the creation of these applications through the proposal of several frameworks and platforms. These tools aim to facilitate the development of context-aware applications, supporting the reuse of components in order to save development time, and hiding technical details by providing an abstraction layer.

However developers have a limited view of the application execution environment, and they can easily overlook some aspects of an application at the development phase that may be important from the user's perspective.

In addition, changes to the application behavior may not be possible at runtime, which threatens to limit the usage of the application. In the other hand, the personalization of the adaptive behavior by the user may not suffice in certain cases and an intervention (e.g. from developers and administrators) might be needed.

The research in context-aware computing should shift its focus to provide more transparency for other roles than the developer, including the end-users, in order to allow the personalization of context-aware behaviors. Also, additional runtime requirements concerning a modification of the structure of the application or a specific internal component should be considered. This would addes the cases where an adaptive behavior has not yet been defined or where the selection of the right action is ambiguous. Furthermore, this will help avoiding usability issues by empowering users as an important actor of the context-awareness.

1.2 Contributions

This thesis aims to facilitate the creation of context-aware applications and extend the range of involved actors to include end-users at different levels like the configuration or customization of the context-aware part of such applications. In addition, it aims to empower the developer in the configuration of the application seamlessly, at different phases of its lifecycle.

This section briefly introduces the key contributions of this thesis through the publications history; Chapter 4 presents a detailed description of the contributions.

Both papers 10 and 11 investigate the motivation for studying context-aware computing, examine the significant research effort conducted in this area, conceptualize and formalize the research problems. Also, they identify the challenges faced by this kind of applications through the examination of how context is used in context-aware applications and where the adaptation control is performed. The contribution consists of the proposal of a new approach for categorizing the characteristics of context-aware applications, as well as providing a new vision

and reference model of context-aware applications.

Papers 3 and 4 cope with the research issues related to context-aware communication services in an enterprise environment: conceptualization, modeling and implementation. Then, they concretize the proposed reference model into a generic and comprehensive framework to guide the architecture design and implementation of context-aware applications that will be used to implement some prototypes of communication services enhanced with a context-aware behavior.

In papers 1 and 8, we propose the design of a framework composed of functional boxes for context management that can be combined by developers to build a custom context processing that best fits the need of their applications. Also, this framework supports the reorganization of the context processing boxes to provide flexibility to developers in terms of context management.

In paper 9, we propose to adapt this for a cloud to address the research issues related to context modeling, description, and its management in the cloud. It aims to provide developers with a scalable cloud-based architecture along with a description language for defining context processing and adaptation procedures.

Paper 7 proposes the conceptualization of context as relational data that can be used to connect entities between them. It introduces a novel concept of the context graph that relies on considering context as a link and entities as nodes. We designed and developed a graph-based framework for managing contextual information and querying paradigm based on graph traversal and exploration.

Another contribution consisted in the integration of new functionalities into the already described framework that aims to empower user by providing user friendly interfaces for defining context-aware adaptation rules that control the execution of services. Paper 6 copes with the usability issues related to mobile applications resulting from the lack of cooperation between these applications, it introduces the design and development of a context-aware service composition framework: composing services to build applications and control their execution with context-based conditions.

Paper 2 and 5 introduce a mobile-based quality of measurment tool that generates a score representing an evaluation of the user experience regarding a specific consumed multimedia service. The tool combines multiple information available on the user smartphone including his/her contextual information in order to personalize the calculated score.

In addition we successfully patented some results of our research work.

1.3 Context

The thesis is jointly carried at department of Unified Communications and Emerging Markets (UCEM) within Orange Labs (the Orange's R&D department

from the telecom operator Orange formerly France Telecom) and at department of Mobile Multimedia Services and Networks within Telecom SudParis. The works conducted during the thesis was supported by two internal research projects within Orange Labs, namely Business Social Communication Services (BSCS) and Work Environment for employees empowerment (WEFEE). The BSCS project is concerned by proposing enterprise collaborative solutions that integrate social dimension and can be accessible through different devices (e.g., PC, mobile, tablets). The contribution consisted of finding new ways for using employees' contextual information to manage incoming communication requests effectively. The WEFEE project has a broader spectrum as it aims to provide employees and self-employed people with an enhanced work environment and corresponding set of tools that can be used inside and outside the work premises. The contribution to this project consisted of the proposal of the context graph as a way for integration dynamical contextual information with social data.

1.4 Outline

This thesis is organized as follows: A first part (composed of chapters 2 and 3) surveys the state of the art and a second part (composed of chpater 4, 5 and 6) exposes the thesis contribution. In chapter 2, we first introduce some scenarios illustrating the benefits of context-awareness and the challenges that face the implementation of such adaptive behavior. Then, we examine the basic concepts related to context-aware computing and well-established definitions among the research community. In addition, we examine the different class of context-aware applications and the core components composing a typical application. Chapter 3 starts by surveying the research challenges related to the support that can be provided to developers for engineering context-aware applications. Then, the chapter reviews a set of relevant works from the state of art regarding the support provided for the engineering of context-aware applications. At the end, the chapter summaries the initial challenges into a set of main requirements that should be addressed and then it uses these requirements to assess the exposed related works.

The second part of the thesis describes the contribution: in chapter 4 we present the main contribution of the thesis, introducing the new concepts for context modeling and processing and the corresponding descriptive model. Then, we present our context management platform that provides developers with multi-level support. In chapter 5, we present the underlying technical details of our context management platform and how developers can use it to get support for the creation and engineering of context-aware applications. Chapter 6 attempts to validate the thesis proposals thorough the implementation of several case studies

while illustrating how the development support provided by the context management platform was beneficial. In addition, the chapter presents the analytical results for performance and scalability evaluation of the platform.

Finally, chapter 7 summarizes the thesis contribution; it also discusses future works and directions.

Chapter 2

What is context-aware computing?

Before addressing the research questions with new solutions, it is important to understand how the existing solutions addressed them and derive the limitations. Therefore, this chapter and the following one survey the literature of context-aware computing. In particular, this chapter introduces the notions that are related to contextual information and gives a comprehensive insight on how context is perceived in the literature. Moreover, the chapter analyzes different classes of existing applications that can exhibit a context-aware behavior, the chapter also describes the application scenarios that the thesis contribution aims to address. The structure of the chapter is as follows: first, section 2.1 starts with the presentation of some scenarios for context-aware applications to expose the need of the adaption to user situation changes. Then, after the analysis of the scenarios, it derives the benefit of adaptation as well as the challenges that could be faced within these scenarios. Section 2.2 defines context, different perspectives in the research community and presents its characteristics and some related notions like situation or quality. Then, section 2.2.3 presents the different operations that can be perfomed on contextual information. The second part of this chapter presents context-aware applications and their classification in section 2.3, introduces in section 2.4, the decomposition of context-aware applications into a set of major functional constituents then based on that it defines an abstract software architecture for the creation of this kind of applications. It also elaborates, in section 2.4.2, on the foundations for architecting and implementing a suitable environment that will facilitate the creation of these applications. In addition, it presents one contribution that consists of a classification method for context-aware applications as depicted in figure 2.2 of section 2.3.1.

2.1 The need of context-awareness

Developers or architects of software are not supposed to be its end user. In addition, the environment under which the software is developed and tested is completely different from the environment on which it will be used. Thus, the initial requirements of the software may be different from the real requirements of the potential users and their environment especially for software with high interactivity with the user. Furthermore, it is very hard to make an optimized configuration of this software so that it will work for every group of users. These challenges are due to many reasons including the non-involvements of users from the first cycle of the software development, the non-understanding of the scenarios for different group of users or the gap between the users' perspective and the software makers' perspective. To tackle these challenges, it is important to guarantee the involvment of the user requirements from the early phase of the development of an interactive system by maintaining a communication channel between the user and the software maker. Such approach for software development is also known as user-centered design.

For certain interactive applications the use of well-established practices for enabling adaptation, like the user-centric design, is not enough as they operate in a highly dynamic environment where a more situational and instant adaptive behavior is needed. The following scenarios are representative for applications that need such kind of adaptations and which the development is enabled by the conceptual framework proposed by this thesis. The following sections then elaborate on the benefit of context-aware behavior and the corresponding challenges that need to be addressed.

2.1.1 Application scenarios

Contextual information are an important enabler for making applications more adaptive and for supporting the user in the accomplishment of complex tasks. In fact, context-awareness is an important part of many studies, in different areas, that aimed to provide adaptive solutions and as a result many frameworks and prototypes were developed. The following scenarios illustrate the use of context for enabling different kinds of adaptation: automatic or manual and enabling a seamless interaction between the "virtual world" and the "real world". In addition, they will serve for the extraction of the main requirements that will guide the development of context-aware applications. Moreover, they will be used later in the thesis as reference use cases for evaluating the proposed concepts.

Scenario 1: the employee tour

In many cases, an employee may not be able to get relevant information or collaborate efficiently with his/her colleagues without the help of an intermediate person (e.g., his manager, office mate). Such cases occur, for instance, when the employee attends a meeting in a different location than his own workplace, has just been hired and joined the company, or moved to another team. A solution is to provide users with an instant access to relevant information about their colleagues and to the objects of their surrounding environment (to facilitate the interaction). For instance, for people walking nearby the system may provide information on their expertise and competencies (relying on intranet directory). It may also locate usual contacts (e.g., contacts from its enterprise collaborative social network) when the user is being at an unfamiliar site of its company. To illustrate how such system can be used, we consider a hypothetical user Sarah who has just joined a new company and received a set of office tools (desk, laptop and a smartphone). This company has just bought a system to enhance the professional communication between employees. On her first day, Sarah takes her smartphone and walks along the offices in the company building. Whenever she meets someone on her way, the system pops up a set of information about this one (name, office n, function, team, expertise, and capabilities, his ongoing projects, etc.) and asks her if she wants to add this person to her contact list. Sarah can also point her phone to an office to get information about the persons who are currently there (just a visitor) or who work there and their status (busy, can be interrupted, etc.). She also can point her Smartphone toward any equipment to get corresponding information. For instance, if she points to a printer she may be provided with information about network name, features, location, how to add it, the name of the technician who must be contacted in case of problem.

Scenario 2: the visitor tour

In this scenario, the hypothetical user Alice is interested to socialize, get informed, and see new things. She usually visits exhibitions to look for job offers, new trend in research or technology. Unfortunately, most of the time, she does not have time to prepare for such opportunities. Once on place, she uses guides available at the exhibition entrance to manually search for information about exhibitors and try to locate them. However, with such method, Alice fails to maximize the benefit she may have from visiting the most important stands. Now, let us enable context-awareness capabilities and see the differences. On her smartphone, a context-aware application is installed to help Alice making her fair visit more interesting and gaining the maximum information. When she gets into the fair entrance, she receives a welcome message on her smartphone giving her a

brief presentation of the Fair, the list of exhibitors, and a map showing all the Fair space and how to reach any exhibitor from her current position. Furthermore, each time Alice uses the system to visit a new Fair; the system saves all data about the trips she made so far. These data will be used for future visits by the system in order to infer which are the most interesting exhibitors for Alice are by matching the exhibitors profile with Alice historical context. In addition, the system provides Alice with a support for starting an interaction (or conversation) with other persons in the Fair. When she approaches an exhibitor she receives on her smartphone a detailed presentation of the company, some information about the personal (name, function) who are present, the exhibited product, service or job, tags or comments from the previous visitors, etc. Furthermore, the system attempts to enrich the environment of Alice by enabling her to point her smartphone (e.g., camera, NFC) to anyone who is walking next to her to get brief information (e.g., function: visitor or exhibitor or guide).

Scenario 3: the technician tour

In this scenario, our hypothetical user is Bob who is a technician working on the maintenance of a complex infrastructure (e.g., oil forage installation, aircraft). To perform his maintenance operation successfully, Bob needs assistance to keep track of what he did, what are the assigned tasks and what to do to fix a problem with a given engine, etc. In addition, he used to move to not suitable place while holding on him only necessary tools with a limited weight. In such environment, accidents may likely happen putting the technician in a critical situation. A smart communication suite is needed to enable the tracking of the technician and for providing him with relevant information at the appropriate time, as well as for linking the maintenance center with the right technician. When providing with a context-awareness assistant, Bob's life becomes a lot easier as many of the tedious tasks will be simplified or handled by the assistant. When Bob starts his tour and enters the infrastructure, the system detects his presence and provides him with a map service that locates him as well as some point of interests (e.g., tasks, machines) within the infrastructure. During the tour, Bob can target with his device any object to get detailed information about it (e.g., name, characteristics, last time being checked, life cycle, logs, and any other relevant information like comments posted by other technicians). After controlling given equipment, Bob can add comments that can help other technicians the next time this equipment is checked. Furthermore, in case of occurrence of an incident in any area, the system shows the incident location with brief description (from error report) in the device of the closest technician having the needed capabilities. As a result, Bob's productivity is significantly increased.

Scenario 4: Health network

Health network (HealthNet) is an interesting example of how context information can be used for an informative purpose. It is a social network that enables its members to share their health related information with other granted members (e.g., family members) or followers. To illustrate how HealthNet can be used by the members of a family, we consider the hypothetical users Paul and Julie. Paul is living in Paris, while Julie his mother is living in Nice. Both use the HealthNet application; they have already created an account. Both have subscribed to access health information of each other. One day, on the road to Nice for visiting his mother, Paul had a car accident and had to enter a hospital at 8pm. His health information will be updated (either by the hospital or by the phone which will sense and detect the abnormality) and the members of his HealthNet are notified. Julie got a notification on her phone with some of the following information such hospital address, his current health situation and doctors. At 9pm, Julie arrived to the Hospital. At the moment she enters the hospital, her phone is notified with the presence of some context-aware services provided by the hospital: hospital map, search engine, etc. At this moment, the system infers that she would probably want to visit her son. Thus the system notifies her phone with the presence of Paul, and sends her more detailed information about Paul (room n, etc.). While these services are offered, the system notifies the hospital stuff about the presence of Julie which has a relation with the patient Paul. Julie uses the hospital map to reach Paul's room. In case the nurse or any member of the hospital stuff wants to talk to Paul's mother (first visitor), he/she is notified about her presence and can join her or start a videoconference.

2.1.2 Scenarios analysis

These different scenarios reference the same context-aware application that consists of providing relevant information at appropriate moment but for different application domains and with different requirements. In the first scenario, the application concerns person-to-person relations and is deployed in an enterprise environment where securing access to information is crucial. In the second scenario, the application concerns also person-to-person relations but is deployed in a public environment where information is publically available with no restrictions. The third scenario is concerned by person-to-object relations where security and real time access to information are crucial. The last scenario concerns person-to-person relations, and the application operates in the frontier between the enterprise and public worlds. As a result, some data is highly sensitive (e.g., patient health information) and cannot be accessed without permissions while no hard constrained are applied on others (e.g., staff or room information). In

addition, the scenarios illustrate the diversity of information that can be used by context-aware applications as well as the different ways of performing a context-aware behavior: manual or automatic mode. In the manual mode, the user holds the smartphone on his hand and interacts directly with an application, i.e., submit commands or visualize information about environment (offices, colleagues, printers, etc.). In the automatic mode, the user keeps the smartphone on his pocket. When an application running in the background senses relevant information it notifies the user (example by a vibration or a ringtone). The user can personalize the information he would potentially be interested to be notified about or the system must infer these information by sensing the user context and historical context information from other users. Example, if another user with similar profile has been interested by something, the user is notified about it. In addition, the scenarios demonstrate the need for a situation specific adaptation and show the variety of information sources used to capture the user situation in order to feed the adaptation decision process. A non-exhaustive list of context information which are needed in these different scenarios include:

- User context: name, function, capabilities, identifier, password, user certificate.
- Object context: network name or url, features, history and life cycle.
- Device context: capability, OS, interaction with user (holding, writing message, phoning), alert mode.
- Location context: place (in field, building, office, meeting, cafeteria), mobility information.
- Social context: relationship with colleague (office mate, manager, assistant, etc.).
- Network context: used and the available access networks and their characteristics.

These acquired information are used in many ways in the illustrated scenarios to provide different kind of services for instance:

Navigation services display map, show paths to reach places,
Information services provide information about objects and persons,
Communication services messaging, social network map, call hold/transfer,
Others services printing, send fax, scanning, search (documents, persons)

The need for adaptation is further investigated in the following subsections.

2.1.3 The context-awareness benefits

Adaptive computing system leverages information about the end user to anticipate his or her immediate needs, offering more-sophisticated, situation-aware service. It is supposed to provide end-users with interactions of an enhanced

quality that exceeds usual one provided by tradition systems. They are assistive by nature (i.e. proactively offer what is assumed to be helpful to the user) and should not distract the user from performing their current task (e.g., consuming most of the available computation resources like CPU and memory). In addition, such systems should empower the user by allowing them to customize the system to meet their needs and preferences regardless of their programming skills. For instance, the system may provide tools and methods for users to identify changes in their situation and manually adjust the corresponding system behavior. This empowerment is benefic for the system adaptation as it involves user's domain-specific expertise in defining relevant adaptive behavior (e.g., override actions, correct decisions). Furthermore, the previous scenarios demonstrate the importance of the calmness property [2]. A calm system can be defined as a system which is able to switch between the periphery and the center of intention of the user. A system is at the periphery when no direct interaction with the user is needed however this does not mean that it is stopped as the case for traditional system but in the contrary the system keeps running in the background to collect context data and process them. In the other hand, the system goes to the center of the user intention as a result of an explicit user request (e.g. to perform a precise task) or as a result of an adaptive behavior that will interrupt the user to provide a valuable service or information without overburdening.

2.1.4 The context-awareness challenges

To enable the previously presented scenarios and provide users with fully adaptive applications, many research and development challenges need to be addressed. In fact, behind these applications is a complex system running in the background to capture user situation by continuously collecting contextual information, discover objects (or services) in the surrounding environment and collect information about them, reason on the collected information to make decision on the right adaptive behavior to expose. Things get more complicated as a result of the highly dynamic nature of the environment surrounding the user where objects (e.g., devices, services) or persons (e.g., colleagues) appear and disappear continuously making the maintenance of the interaction with them a challenging task. Furthermore, this dynamicity brings new challenges for the adaptation process as some application usage scenarios that took place at runtime may potentially not been anticipated at the application development phase. The different challenges that can be derived from the previous scenarios can be categorized into technical and non-technical challenges. Both are further investigated bellow in the following subsections.

A. Technical challenges

This category of challenges gathers a list of the technical issues [3] that face the implementation of such systems:

- User Intent: a context-aware system has to track user intent in order to deliver services that fit it. Thus, user intent has to be represented internally, inferred with an acceptable accuracy;
- Cyber forging: for certain services, the user's mobile-device computing resources have to be augmented with an existing wired infrastructure. Thus, a mechanism for mobile-device presence detection is needed, it is important to have a certain level of trust between the different devices; load-balancing mechanisms may be needed, etc.
- Adaptation strategy: in case of the demanded resources exceed the offered, adaption strategies (like reducing resource utilization, resource reservation, suggesting an action to the user) have to be deployed. The problem is then transformed into how to choose the adequate reservation, when to use one of the strategies;
- Energy management: sophisticated services are energy consuming; how to asses and control the consumed energy for a giving service;
- Client thickness: does a context-aware service demand a powerful user device?
- Balancing Proactivity and Transparency: proactive actions may not coincide with user expectations and turn to be annoying. Thus, it seems that sometimes the system has to ask the user for performing an action. When to use one of the strategies, how to consider user preferences;
- Privacy and Trust: in terms of authentication, authorization, etc.
- Impact on Layering: context-aware services need to merge information of different level of abstraction. This merging may have performance impact: how layers are created and used for delivering a certain service;

In addition to these challenges, acquiring user feedback is another challenge to consider [4] especially from the user interface perspective, i.e., how to present to the user an adequate mechanism to allow him providing his feedback. In addition to the consideration of this feedback in the system adaptation process for instance as user's preferences.

B. Non-technical challenges

In this category of challenges fall the challenges that face the definition of an adequate business model for context-aware systems. A business model is a description of how a network of organizations co-operates in creating and capturing value from technological innovation. Previous business models [5] focused

on the actors, relationships, and value object exchanged, with less attention to cross-company collaboration in complex value networks – such as context-aware systems. The authors in [5] presented the challenges facing the design of business models for context-aware systems as:

- The definition of a compelling context-aware services for particular consumer segments, taking care of privacy issues and creating trust in the service;
- The integration of emerging technology platforms;
- The combination of multiple revenue models;
- The division of roles in a complex value network, including new business roles.

In addition to the present challenges, the development of adaptive systems that address the previous scenarios may face other kind of challenges like regulation-related challenges [6] due to the very intrusive nature of these systems. For instance, some countries have laws that forbid employer from gathering personal information (and thus contextual ones) about their employee unless there is a specific circumstance and for a limited time. Also, there may be a law or a company-specific rule forbidding their drivers from answering calls while driving. Moreover, its important to building trust between the company (who deploy the system) and its employees (who you benefit from the offered service) is needed to make these ready to share their contextual information.

2.2 Contextual Information

In the literature, contextual information are perceived primarily as information about the user himself like:

- Identity that characterizes him/her with an explicit and unique identifier within the domain space of the application,
- Location that corresponds to geographical data and spatial relations (e.g., positioning data, orientation, regional relations to other users/objects or neighborhood).
- Status that refers to physical factors like, vital signs, tiredness, emotional state or the current affiliation.
- Relationship that describes the relation between users like friendship or relation between the user and objects on its environment like ownership.

It also concerns environmental information which are mostly physical measurements (for example, the current temperature, the ambient illumination or the noise level) in addition to settings information that may be used for configuration purpose like privacy policies (controlling how, where and why information and content are being accessed by systems and devices). Besides these data which

are manually or automatically collected, contextual information may concerns derivative knowledge that describes user's activity, behaviors, needs and intentions. For instance ontology-based techniques can be used for this generation as they are able to understand information through the use of vocabularies.

2.2.1 Definition

Context, as a term, is widely used in many research disciplines like linguistics or archaeology. In Computer science, it is argued [7] that context represents all information intrinsic to an entity (e.g., user, device) that can be acquired, made explicit and published to applications, in order to enable them to adapt their behavior to the entity's state. Context data may not intervene directly in the interaction between the user and the application but instead serves as constraining information. This general and vague definition of context introduced by Dey et al. [7] is further interpreted in two different viewpoints by the community of researchers in the context-aware computing field. Another definition by K. Henricksen [8] *Context information is a set of data, gathered from sensors and users, that conforms to a context model. This provides a snapshot that approximates the state, at a given point in time, of the subset of the context encompassed by the model.* This definition provides an explicit separation between the notion of context information and the notion of context modeling, however it lacks the description of context-aware applications as consumers of context information that provide adaptive behavior.

Theory and Practice

From the industry viewpoint [9] (e.g., travel or retail companies) context is about location, identity, and state (physical, social, emotional or informational state) of people, groups, and objects. This information is acquired to answer questions like for example:

- Who is using the service? What are his/her preferences or habits?
- What is doing the user? What are his current activities?
- Where is the user? What is the logical or physical place where the service is invoked?
- When? At what time are his/her actions occurring?
- How? Which device is used to access the service?

With this in mind, context-awareness for industries is defined as the ability of a service to adapt its response to user's requests depending on whether the service is accessed from a mobile phone or accessed from a laptop, on his/her identity and his/her role (e.g., particular or professional client), from where the user is looking for the service, etc. However, most existing commercial context-aware

applications (e.g., Socialight, Nike+, Foursquare, etc.) are restricted to the consideration of location and identity as the primary types of context.

From the academic viewpoint [10] a broader definition of context is consider and which goes beyond location, presence, and sensor-based information. Context is about facts, rules and axioms that can be used to describe a state of an entity (e.g., user, device) at a given time. Context-awareness is considered as a special kind of formal logic systems on which well-established Artificial Intelligence theories and algorithms (e.g., rule-based inference) can be applied for automating the processing of inferring new knowledge and reasoning on facts representing user situation.

In this thesis, while keeping in mind the academic viewpoint, we intend to take into account the industry viewpoint that is usually less investigated.

Situation

The set of contextual information sharing the description of a single user can be compiled into a General Human Profile (GHP) [11]. This profile is filled by the whole contextual information that can be captured from different sources and integrate them into one structure. The access to this structure is exposed to third-party applications in a unified way, however only a subset of these data will potentially be used by a given application.

Figure 2.1 depicts an example of a GHP that gathers most potential contextual information that can be considered for an adaptive communication service.

A snapshot of the GHP structure at a given instant of time represents what is called situation. Adaptive applications continuously perceive the user situation until a match is found with a condition of interest then the adaptive behavior is triggered. The completeness of the perceived situation regarding the overall GHP depends on the need of the application as the location and identity information are sufficient for many cases.

2.2.2 Properties

Contextual information shows a set of characteristics that needs to be carefully considered by context-aware applications in order to effectively use and manage the contextual information. These properties are further discussed below: Contextual information are dynamic by nature, the measurements change over time (e.g., temperature, location), with a frequency of change that is not the same for all contextual information. For instance the user location changes very frequently, however the birth date is a static information that does not change. The result of this dynamicity is a challenge for the storage of context as their persistence (the amount of time during which the information stay unchanged) cannot be

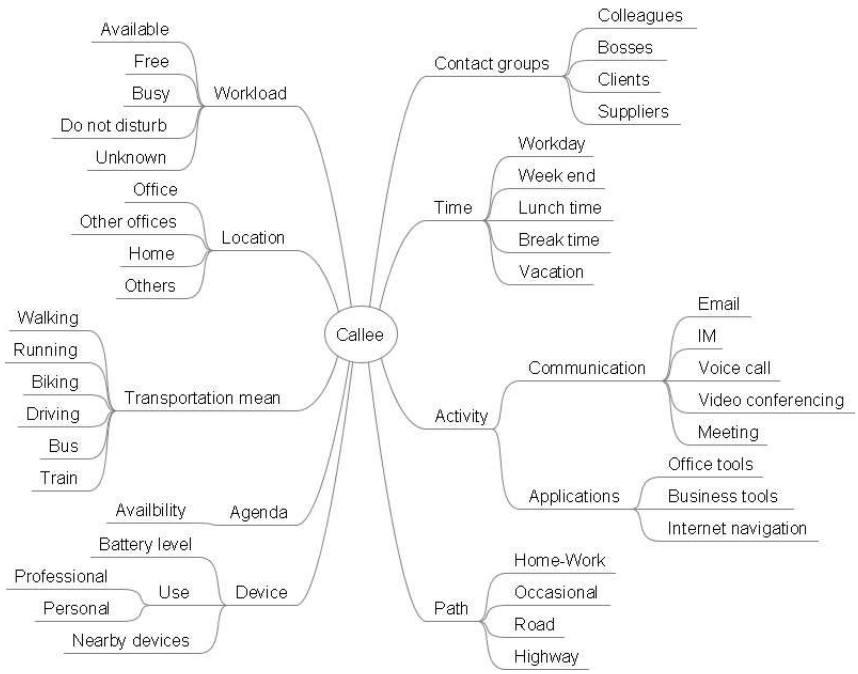

Figure 2.1: Example of context data

handled in the same way. Also, context exhibits the imperfectness characteristic because the context acquisition operation (i.e. sensing) may fail or the source of the information itself is imperfect (e.g., the sensing environment may be noisy). In addition, context information that describes the situation of a given entity is mostly to be interrelated. This interrelation may exist between the same kind of data (typical example is location: the Global Positioning System (GPS) positioning information derive the civil address information) or different kind of information (e.g., the velocity depends on location and time). These dependencies are due to the process of transforming context data to some knowledge of interest to the application via the use of techniques like interpretation, aggregation, derivation, etc. Furthermore, the context is heterogeneous as it is collected from different sources. For instance, it can be collected from the user (e.g., via asking him/her to fill a form), sensed from a device (e.g., via physical sensors), profiled from continuous observations (e.g. monitoring the communication logs to find the closest contacts) or derived (e.g. from agenda and location information infer the activity of the user). Integrating such various type of information is challenging for making reliable adaptive behavior. Moreover, the level of abstractions or granularity of contextual information is a variable as it depends on the type of the data source. For instance, an ambient light sensor may provide

state information on ambience like low, medium or high whereas another type of sensor will provide numeric measurement on a defined range of example a value between 0 and 10. This granularity concerns the abstraction of the same kind of information; another type of granularity concerns the abstraction of two or more different kind of information which is also called the derivation process. An example of derivation is for example the use of data from a compass and an accelerometer sensor to derive the direction information. These characteristics which are unique to contextual information make the use of such data a challenging task. They imply the need for augmenting context, along the process of their management, with meta-information describing their quality (for example: precision, frequency, confidence) to support their management.

2.2.3 Related operations

The systems involved in the management of contextual information can be represented as a process composed of a set of interrelated tasks that takes as input an array of contextual information to transform them to a meaningful knowledge as output. The latter, is then used by third-party applications as a triggering input by its internal adaptation engine.

The different tasks are complementary to each other and not overlapping to cover all operations of the context process and keep the dependency among them as much cleanly as possible. In case a given task is non-appropriately designed this will affect the other tasks depending on this one and as a result affect the whole performance of the context process. Furthermore, the complexity of the representation of context data may vary dependently on the operations a given task may need to perform.

This section discusses how context-aware systems work by decomposing them to the major functional steps of the context management process. In addition, it will describe the different possible data structures and operations performed by each of the steps to guarantee a manipulation of context data in an efficient way.

Acquisition

This tasks aims to observe the user and collect knowledge on his situation. Different approaches can be used to supply context data to the whole system and can be classified into: explicit approaches where the user is involved or implicit approaches where the collection is performed by specialized software component in the background without any interaction with the user. The two approaches are described in more details as follow:

Explicit context collection example of collection techniques of this type includes the use of forms or questionnaires as means of capture information

describing some of user properties (e.g. preference, interest).

Implicit context collection are mostly used to collect, without a direct interaction that involves the user, physical properties that concern the user himself (e.g. location, heart beats) or his/her environment (e.g. temperature, lightness) through dedicated software components called sensors. In addition, the collection can be performed by querying static sources of information like databases, files or web services (e.g., weather forecast).

The selection of the appropriate collection approach depends on the kind of application and on parameters like reliability, or sampling rate (i.e. interval of time between two samples) that the collection method should guarantee. For instance, a GPS sensor is more reliable than a location information provided explicitly by the user, furthermore it can be used to acquire user location at any frequency while requesting the user regularly can become an overwhelming task. However, for privacy considerations it may not be suitable to automatically collect such sensitive data. In the other hand, the interpretations or assumptions made on automatically collected context data may not match the real situation of the user. For instance, if we automatically collection location data to infer how the user is moving, then we may interpret the moving speeding as the user is running while he may be biking. Thus, the selection of the right context acquisition method should be carefully considered and needs a good knowledge on the different aspect of the application.

Processing

This task aims to generate more insightful knowledge based on combining and processing available context data. This task is important as for some applications the information acquired by the previous task may not be enough to perform an accurate adaptive behavior as they need more deep analysis and interpretation. The context processing is a complex task as it involves many kind of information (e.g., sensors measurements, user explicit inputs) potentially represented differently, thus different techniques may need to be used. Following is a non-exhaustive list of widely used techniques in knowledge engineering [12] that some have been successfully applied to context processing:

Statistical models widely used mathematical approaches that aim to formalize the relationship between stochastic (not deterministic) variables thanks to mathematical equations. There exist too many statistical methods like time series models, linear models, regression, etc. Not all of these methods can be appropriate for application on context data, thus model comparison techniques like the estimation of the mean squared error should be applied on the dataset to choose adequate model.

Data Mining kind of more advanced approaches that go beyond simply modeling a data set by trying to find some previously unknown and interesting patterns then transforming this knowledge into general rules. Example of purposes where data mining approaches are useful includes the analysis of groups of data records (cluster analysis), unusual records (anomaly detection) and dependencies (association rule mining). Context-aware recommendation systems [13] are examples of using data mining techniques.

Expert systems aim to imitate the ability of humans in decision-making in solving complex problems by reasoning about knowledge and not by following a pre-defined procedure as is the case for conventional computer programs. They can be divided into an inference engine which are programs able to reason on rules expressed in a given logic (e.g. propositional logic, first order predicates), and a knowledge base which contains a collection of IF-THEN rules expressed in a natural language (e.g. IF it is living THEN it is mortal). The reasoning of such systems consists of running the engine on the facts available in the knowledge base. The ubiES [14] system is an example of works that attempts to use expert systems as the reasoning engine to deliver context-aware behavior.

Artificial Intelligence gathers a collection of approaches that aim to bring a sort of intelligence (i.e. deduction and problem solving, knowledge representation, planning, etc.) to computer programs. Some of the approaches developed by Artificial intelligence (AI) includes: logical approaches like Fuzzy logic, probabilistic methods for uncertain reasoning like Hidden Markov Models, neural networks, etc. An example of the application of AI techniques for context-awareness is MoBe [15]. This system provides an infrastructure for supporting context-aware mobile applications development and specifically the inference on context data through providing different built-in AI techniques (rule-based systems, Bayesian networks and ontologies).

These different derivation techniques rely on the quantity and quality of the available data to generate interesting knowledge. The community of researchers in Artificial Intelligence developed robust systems for automated inference that most of the time relies on a knowledge base, in the other hand community of statisticians developed tools for inference from a corpus of data. The quality of inference in the former relies on the quality of the content of the base and the quality of the latter relies on the quantity of the data initially held. As a result, a careful selection with a good understanding of the properties of each method should be done.

Representation

Each of the previous processing techniques relies on a corresponding modeling approach for representing the data (or the rules) which makes the co-existence of different processing techniques for a single application a hard task to fulfill. For example, statistical models needs only the data that can be stored in arrays while a rule-based system needs to represent both the data (e.g. using ontologies) and the inference rules (e.g. using RuleML[1]).

In some cases, it is the choice made for the modeling approach that derives the choice for the processing technique. It is the case for modeling approaches (e.g. ontology-based) that embed the semantic aspect of the data along with the data itself. This coupling is highlighted in the following paragraph by classifying the different context modeling approach proposed in the literature into two categories based on the level of semantic constraints they imply to the processing techniques.

Models with strong semantic constraints Resource Description Framework (RDF) is a language for describing tagged oriented graphs. It allows describing triples (subject, predicate, and object): the subject is the described resource, the predicate represents a property type that can be applied to this resource, and the object represents data or another resource. Each triple corresponds to an oriented arc tagged with the predicate, the source node is the subject and the destination node is the object. Based on RDF, Web Ontology Language (OWL) is an interesting context modeling and reasoning approach for building context-aware systems as it provides support for complex relation representations and knowledge inference. With OWL, knowledge inference is performed by rule languages like Semantic Web Rule Language (SWRL) or Semantic Query-Enhanced Web Rule Language (SQWRL) [16]. In [17], the authors used OWL to model context with more than 50 hierarchical ontology classes; SWRL and SQWRL were used for inference of high level knowledge from acquired context data. Limitations of OWL-based approaches, as stated by the authors, include the difficulty of reasoning on historical context (e.g., the different locations that a user visited the last week). Also, OWL-based rule languages are not flexible enough to support complex queries (e.g., calculating total duration, from historical context, of the time that a user had spent in a specific location), and have performance issues.

Models with weak semantic constraints Model Driven Architecture (MDA) is a powerful approach for system development, as it is a model centric and flexible development process providing an automatic application generation and support for application maintainability. In [18], the authors pro-

1. http://ruleml.org/

pose ContextUML for modeling context provisioning and mechanisms for context-awareness. In [19], the authors propose an Unified Modeling Language (UML) context modeling approach for extending Business Process Management (BPM) to enable a new queryable model for BPM artifacts (e.g., selecting services based on their context, display tasks on map instead of role-based list). eXtensible Markup Language (XML) is a widely used standard markup language for representing any kind of information in a human and machine readable way. It is also used for information exchange by many communication protocols (e.g., eXtensible Messaging and Presence Protocol (XMPP)). ContextML [20] propose an XML-based model for representing and exchanging context. It allows the categorization/grouping of context into scopes (e.g., activity, settings) related to an entity (e.g., device, user). An entity (e.g., user) can have many scopes (e.g., location, activity) of context. A scope (e.g., location) gathers some closely related context data (e.g., longitude, latitude, civil address).

Adaptation

This task is in charge of deciding the appropriate adaptive behavior to perform as a result to context changes. It represents the main task of a context-aware application and the component responsible for executing this task should hold appropriate knowledge for identifying the right behavior for the right situation.

In adaptive applications [21], the decision knowledge is commonly expressed as rules (e.g. IF-THEN rules) due to their expressiveness in describing adaptive behaviors, the simplicity of their implementation and integration with the application business logic. In this particular configuration, the decision regarding what behavior to expose results from the interpretation of the rules with a consideration of context data. The components of rule systems are a set of rules of the form of "IF-THEN" and a fact base which consists of the input information of this system. The rules have two sides, the left one consist of a Boolean expression and the right side gathers a set of applicable actions. A rule system execution follows the match-resolve-act cycle [22]:

- Match: in this phase, the preconditions (i.e. rule's left side) of the existing rules are matched against the fact base, from every satisfied rule the action (i.e. rule's right side) is added to the set of applicable actions.
- Conflict-Resolution: the generate set of actions may potentially contains conflicting actions; this second phase consists of choosing best candidate actions for execution.
- Act: finally, the selected actions are executed and the reasoning process may return to the first phase for a second loop or wait for a triggering event to restart the cycle.

The Munich Reference Model [21] [23] proposes four main operations representing the steps of an adaptive behavior: triggering the adaptation process, finding appropriate adaptation and resolving it to concrete actions, and the execution of an action. They are described as follow:

- Trigger: after identifying a need for adaptation, the system triggers the whole process that will result in performing adaptive behavior,
- Find: after the process is triggered, the system looks for the appropriate behavior (which represent an abstract action) that match the conditions of the current situation,
- Resolve: the select behavior is resolved to concrete actions that can be executed,
- Execute: the action is then executed either by the system or proposed to the user in case it should be executed by him.

More generally, the execution of the precedent steps of the adaptation process is influenced by the following set of information:

- Information about adaptation: includes knowledge for the determination of a need for adaptation as a result of the identification of context changes (e.g. entering a specific situation). In addition to information that helps determine the relevancy of context changes for a given application. Information regarding the quality of context (e.g. changes frequency) may also influence the selection of the appropriate behavior.
- User involvement: Involving the user in the adaptation process aims to motivate him to use the adaptive property of the system. This requires knowledge that would regulate the alignment of the system behavior resulting from the adaptation process with the user expectation. Also, it needs information that describes the degree of control of the user on each step of the adaptation process.
- Information about how each step of the adaptation process is operated whether automatically by the system, by the user or in an assisted mode where the system helps the user in performing the action. For instance, complex and/or repetitive actions are left to the system, while in case of uncertainty regarding the best action the user may be inquired.

Actuation

This task is in charge of executing the actions resulting from the previously selected adaptive behavior. The executed actions may affect directly the user and be observable for him or may be performed in the background and as a result they are not perceivable by the user at least at the execution time.

In context-aware applications, traditional information input approaches (e.g., mouse, keyboard, gesture) are expanded by other information acquisition ap-

proaches like the use of sensors. Similarly, traditional information output approaches (e.g., screen) can be expanded to any actuator device that can act on the user environment (e.g., mechanical engine, light controller, etc.). For instance, a notification can be displayed to the user on his screen, by playing a ring tone or lighting a light-emitting diode (LED). More generally, the properties of context-aware application that can be expanded as a result of their context-awareness include the set of HCI methods used to input data or commands to a system, and the methods used for outputting data. In addition to the content presentation, i.e. how the system present an output data to the user.

Operations dependency

The previously described context-related operations are not independent; in most cases the context acquisition is performed with a precise idea of how the acquired contextual information will be used later by other operations (e.g., for inferring new knowledge). Thus, it is important to understand these dependencies as well as the kind of dependency between each subsequent operation. Few research works addressed this question, for instance in [24], the authors identified two main types of dependencies application-context dependency and application-application dependency. The former is the result of the need of the application for context in order to perform its main task. The latter is the result of application depending on another.

In the general case, the dependency between the different operations is influenced by the flow of knowledge in addition to the flow of control deriving from circulating information among these operations.

At which time of the context-aware application lifecycle the knowledge is generated by a given operation is an important factor for drawing the dependencies. There is two main moments in the application lifecycle: compile time (moment of the application design) and runtime (moment of the application operation). The knowledge generated at compile time is embedded with the application prior to its operational time. This knowledge holds the business logic of the application and it is supposed to live unchanged as long as the application is running and may eventually be modified on application updates. It includes the model of the different entities (and their interactions) that will interact with the application, the context acquisition and actuation methods, derivation and inference methods, and adaptation rules. The knowledge generated at operational time holds information on how the application is interacting and perceiving its environment. It includes the acquired context information, for example the context history which is collected progressively to be processed later by long running batch operations for interpretation and generation of additional knowledge.

Another factor for the dependency between the operations is how updates

(e.g. addition, deletion) to the information (or knowledge) retained by an operation may propagate to modify the information retained by subsequent operations. For instance, the GPS location obtained by the acquisition operation is then abstracted to high level information (e.g. street name) by the processing operation or used the actuation operation to update the location display in a map.

Furthermore, another source for dependencies is the how the information is represented as this may limit the kind of processing techniques that can be used to generate knowledge or decisions. In addition, for exchanging this information between two operations, an intermediate translation step is needed to convert the information to a representation understandable by the subsequent operation.

2.3 Context-aware applications

To be able to understand the operational aspect of context-aware applications it is very important to analyze and carefully examine existing applications on different application domains. An illustrative scenario for a context-aware application taken from a specific application domain was already introduced in a previous chapter. Following is a non-exhaustive list of context-aware applications taken from different application domains:

Communication Socialight [25] aims to allow users to post messages so that it will be delivered to anyone who is located nearby. Calls.calm [26] allows users to communicate information about their situation, as well as the communication channels they are available on, to anyone willing to contact them.

Assistant SECE [27] uses multiple context information to enable the user to write adaptation rules for managing the behavior of a communication service. OnX [28] a mobile assistant that uses multiple context information to trigger the execution of an action.

Tourism Use visit history to guide the user toward new artworks to discover [29] or use information about the artwork the user is seen to present in convenient way relevant information about it [30].

Fieldwork Capture information (e.g. location, problem description) about incidents to provide it to adequate technician; for instance who is located nearby and who may have required skills. Capture and collect environmental information (e.g. weather, signal strength) and attach them to a location in order for instance to visualize the aggregated data in a map [31].

Shopping Enhancing the user shopping experience by providing details on products, helping the user to locate him/her and to find his way to a given product [32].

Museum Combine physical and virtual context to assist the visitor by adapting the provided information about artworks based on their interests and knowledge [33].

The different works described in the previous table relies on some common functionalities of context-aware applications and expose specific patterns related to the usage of contextual information to influence the adaptation process of the target application. The derivation of the different types of context-aware applications can be obtained by clustering the properties exhibited in the earlier examples in order to understand how this kind of applications are functioning as well as their main building blocks.

Many categorizations of context-aware applications have been proposed in the literature. Some were concerned about how contextual information are used by an application as well as the role of taken by context, others focused on the functionalities provided by the context-aware application. In this section, we will present both classes of categorizations.

2.3.1 Context-centric categorization

Contextual information can be used in many different ways. Basically, context can be used passively just by displaying it to user, actively in a filtering mechanism to select best suitable choice or used for triggering some actions when the context changes. In [34], the authors studied some basic 3GPP (3rd Generation Partnership Project) telecommunication services and proposed a segmentation of context-aware applications based on how the user context is used: whether context awareness is used to modify the service mechanisms, or used to reengineer the services in a more creative way. The different classes of context-aware services resulting from combining these two axes are as follow:

Parametering most parameters of telecommunication services of the 3GPP [35] specifications are defined by the carrier at the user subscription without involving the end user. An example of such parameter is the no answer timer of the Call Forwarding on No Reply (CFNR) [35] feature which corresponds to the time the phone should ring before forwarding the call (for instance to a voice mail). In this case, the CFNR feature can be made more adaptive to the user situation (e.g. to the type of caller) in order to appropriately handle the call (e.g., forward a professional call to a colleague).

Triggering some services of the 3GPP specification are always on or run on user request. This behavior can be adapted to the user situation to invoke the service only on appropriate time. For instance, the Call Deflection (CD) feature that enables calls to be deflected to another destination in order to not interrupt the user with incoming calls is triggered on the user request.

This feature can be made sensitive to the user situation in order for instance to be triggered when the user is driving a car.

Abstracting some services can be aggregated by abstracting the conditions that they rule the behavior of this service. In fact, some forwarding services from the 3GPP standards (Subscriber Not Reachable, Not Logged-in, etc.) rely on a single separate context dimension that when combined can create a unique service.

Reinventing the idea would be to re-think an existing service with context-awareness in mind. For instance, an always on service that handles some predefined calls as important will be made more intelligent if it dynamically assesses the importance of the call (e.g., by recognizing the subject of the call by analyzing the speech) and act dependently.

Other possible alternative classification criteria could concern the timeliness of context exploitation, the location of the intelligence responsible for making adaptive decisions, or the type of the final behavior exhibited by the system.

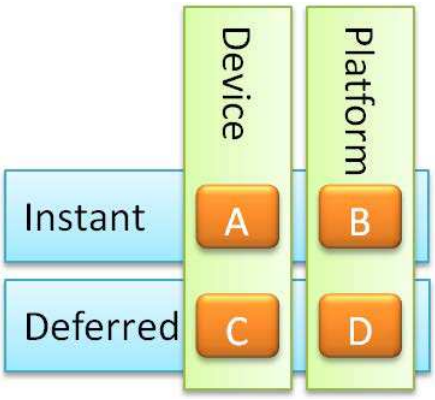

Figure 2.2: Two axes segmentation of context-aware applications

In [36], we propose a classification based on two axes (as depicted in figure 2.2): how the sensed context is used, either instantly or in a deferred way, and where the adaptation is performed, either on a service platform or on an end-user device. Combining the two axes would give us the four following cases:

– Case "A" gathers the applications that exploit instantly the sensed context to perform the adaptation on the end-user device. An example of such application is the Rendez-Vous service that aims to help a group of people to organize a meeting by sensing the calendar data of each one to check their availability in the future. An example of earlier works of this class is Calls.calm [26] that enables the caller to be aware of the callee's situation

(which is manually updated) in order to decide if it is suitable to initiate a communication or not.
- Case "B" gathers the solutions that exploit instantly the sensed context to perform adaptation at the platform level. An example of such solutions is the location-based messaging service Socialight [25] that aims to allow users to post messages so that it will be delivered to anyone who is located nearby.
- Case "C" gathers solutions that exploit in a deferred way the sensed context to adapt the interface on the end-user's device. A typical example is the adaptive address book in which contacts within a contact list are sorted based on their relative importance (e.g. professional contacts shown first during working hours). Another example is SocialFusion [37] a recommendation system that integrates diverse contextual information (e.g. social and mobile data) to generate a group-based or individual recommendations (e.g. displaying recommended movie trailers based on preferences of the users jointly watching the same display).
- Case "D" gathers applications that use context in a deferred way (i.e., by processing historical context data) at the platform level. An example of such applications is the works conducted by K. Hamadache et al. [38] where a communication management service was proposed. This service uses historical information to assess the importance of the caller to the callee, this knowledge as well as other information (e.g., presence) are used as part of a set of predefined IF-THEN rules in order to handle an incoming call appropriately. An example of such rules would be if the callee is busy and if the communication is not urgent, then redirect the caller to the callee's voice box.

An example of works from class "A" includes the Citron framework [39] for acquiring and processing contextual information from personal devices. The authors use a prototype of a user device, called Muffin embedded with multiple sensors classified into environmental sensors (e.g., air temperature sensor, relative humidity sensor and barometer), biological sensors (e.g., alcohol gas sensor, pulse sensor, skin temperature sensor, skin resistance sensor), Motion/Location sensors (compass/tilt sensor, 3D linear accelerometer, grip sensor, ultrasonic range finder, GPS), and other sensors (e.g., RFID sensor, front/rear camera and microphone). The acquired context is used to track the user's state and path, and then display the pathway in the device's screen. An example of works from class "B" includes INCA [40] a context-aware communication assistant that uses callee preferences and context to handle communications (e.g., forwarding calls). This assistant relies on a layered approach in order to support adaptive context-aware communications. The main layers are session layer for signaling protocols, communication layer that include the environment of the communication (e.g., callee/caller

identifiers, networks and devices), and user layer that models users' preferences and profiles to provide personalized services. Each layer (e.g., communication) holds an internal state which is updated after the reception of events from the lower layer (e.g., session) and the execution of actions requested by the upper layer (e.g., user). An example of works from class "C" includes a context-aware mobility management system for mobile phones [41] which is able to manage the connectivity of mobile applications in a transparent way by deciding which interface to use at a certain time and performing horizontal handovers between two network access of different technologies (e.g. WiFi and WiMAX). It can use different contextual information like network interface metrics (e.g. received signal strength), device characteristics (e.g., battery, memory), GPS coordinates, etc. The system uses statistical machine learning techniques to generate from sensed contextual information predictions about availability of network interfaces, and then it decides when performing handovers. Predictions are made possible due to temporal and spatial patterns of user's daily behavior (e.g., taking the same way to go work every morning). In class "D" we can find works like the Reality Mining project [42] that aimed to extract new knowledge from data acquired from user's mobile devices. This project performs a study of human activities to recognize recurrent social patterns by analyzing a dataset of phone's contextual information of 100 students stored during the academic year. The recognition of recurrent patterns is with big benefit because it makes possible the automation of certain actions.

2.3.2 Functionality-based categorization

A logical way to segregate context-aware applications is by considering the purpose of the application itself or the kind of functionality it provides to the end-users. In [43], the author presented different classes of context-aware applications which are listed below: Tracking Services [44] that aim to track the exact location of persons (e.g., children, friends or family members) or objects (e.g., vehicles, smartphones, etc.). Examples context-aware applications that belong to this class include: fleet management [45], location-based messaging service (e.g., Socialight [25]), etc. Navigation Services aim to guide a user or an object from a starting point (e.g., current location) to a target destination. We can further divide this class of applications to the sub-classes of blind guidance or assistive guidance. In the first sub-class, the user plays a passive role and let him-self be routed by the application to the target service he/she is looking for (e.g., nearby gas station, ATM, restaurant, etc.). LoST [46] (A Location-to-Service Translation Protocol) is a typical Internet protocol that can be used to locate the target service. In the second sub-class, the user plays a more active role as it knows exactly his/her target destination and thus uses the application as an assistive

tool to reach it. Examples of such applications include indoor guidance in exhibitions [47], or the intelligent routing of vehicle traffic [48]. Information Services that aim to provide users with relevant information that best matches his/her situation and needs. Most common kind of these applications are location-based that provides, or simply display, information (e.g. weather, tourism) which are relevant to a given location. In [49], the authors presented an example of such kind of applications that aims to retrieve information from a repository, which is filled from heterogeneous data sources, based on the user preferences and context data, as well as on information about the process workflow the user is involved in. In [50], we presented an application that intends to integrate information about employees and there in-between relations as well as information about membership to communities in order to facilitate the discovery of potential contacts or relevant documents. Communication Services that aim to leverage the complexity of establishing and managing communications in a way that goes beyond the sample use of the manually edited presence information to handle calls. Typical examples of applications of this class could help the user to handle incoming calls on his/her behalf or help the caller reach someone who satisfies some specific criteria (e.g., who have some specific expertise). In [51], the authors presented a call routing system for enterprise customer care service that facilitate the establishment of communication between a user and an expert agent based on the latter's contextual information: location, activity, availability, communication channel and expertise. Entertainment services are kind of services that holds the attention and interest of a user through leisure activities (e.g., music, games), communications (e.g., social media, instant messaging, Twitter), or commerce activities (e.g., shopping). In [52], the authors proposed to embed an advertising display system into vehicles (e.g., cabs, buses or planes) instead of using the outside space for advertising purpose. The system offer users a various kind of context-aware information (e.g., advertisements, points of interest, events, etc.) during a cab ride. In addition, it allows advertisers to upload advertisements contents and define the corresponding areas where it should be displayed. In [53], context-aware in-flight entertainment system that provides plane passengers a personalized service to reduce their stress level (mostly due to long distance flights) by intelligently choosing an entertaining service (e.g., game) based on the passenger personal demographic information, activity, physical and psychological states.

2.4 Designing context-aware applications

The type of contextual information ranges from user information (e.g. activity, mood) to social information that describes a relationship (e.g. colleague,

family member, friend, etc.). Furthermore, they may concern network parameters like quality of service (e.g., round-trip time), or device related information (e.g., battery, connectivity).This diversity makes the management of contextual information a challenging task, and as result the development of such applications becomes a complex problem. A well-known approach for building such complex systems is called Modularity [54] which consists of decomposing the whole system into smaller subsystem that can be developed separately and reused later in other systems. For instance, the management of context can be separate into specialized building blocks responsible for the collection of contextual information with the use of sensors (e.g. calendar, light, battery charge, etc.), the modeling of context that can be anything (e.g., GPS location or a street address, time, etc.) and reasoning about it to produce an adaptive behavior (e.g., automated call transferring, the proposal of a meeting session, etc.). The remaining of this section describes further the building blocks of context-aware applications and how they can be efficiently designed through a layered architecture where each layer is specialized in the implementation of one unique function. In addition, it discusses the dependencies between the different layers.

2.4.1 Layered Architecture

In the literature, the development of context-aware applications is supported through the use of Context Management Systems (CMS) that implements the different operations described earlier. The support is provided through exporting functionalities (e.g., via Application Programming Interface (API)) that can be invoked by context-aware applications. From the conceptual viewpoint, CMSs are mainly based on the Producer-Consumer design pattern [55] where context sources (e.g., sensors) play the role of Producers, and context-aware applications play the role of Consumers. From the implementation viewpoint, CMSs can be classified into centralized or distributed architecture as depicted in figure 2.3. In the centralized architecture (figure 2.3.a), a central point often called the broker [56] is introduced between the producers and consumers. All context requests are handled by the broker, which forwards it to the right component. In the distributed architecture (figure 2.3.b), a point-to-point model is used where each pair can be either a producer or a consumer, in some cases a pair may play both roles. In the first architecture, producers and consumers are decoupled from each other and the communication between them are handled through the intermediate broker, while in the second alternative, the different components have to know each other (e.g., by regularly sending multicast or broadcast messages for announcing themselves), like in the middleware-based CMS [57].

From the functional viewpoint, a layered framework [58] [59] (as depicted in figure 2.4) can be used to represent context-aware systems from bottom to up by:

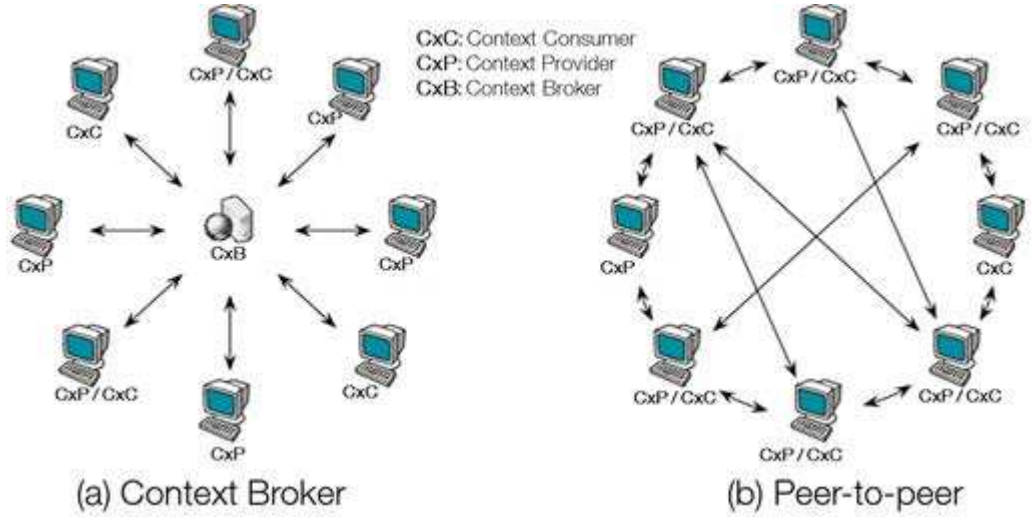

Figure 2.3: Alternatives for distributing context-aware applications components

sensors layer, raw data retrieval layer, preprocessing layer, storage/management layer, and application layer. The CMS is responsible of retrieving raw data from sensors, abstracting and combining the sensed data into high level context, and then of making it available for context-aware applications. The first layer (Sensors) is a collection of sensors responsible of retrieving raw data from the user environment (e.g., user device, social network, or used access network). Context sensors can be classified into:

- Physical sensors or hardware sensors that are able to capture physical measurements like light, audio, location, temperature;
- Virtual sensors that are able to sense data from software applications or services (e.g. sensing calendar entries);
- Logical sensors that are able to aggregate information from different sources (combine physical, virtual sensors with additional sources like databases) to perform complex tasks.

The second layer (Provide layer) makes use of specific API or protocols to control components of the sensor layer (i.e. sensors) and to request data from these components in a synchronous way through direct calls or asynchronous way through subscriptions. These invocations should be as far as possible implemented in a generic way, making possible to replace sensors (e.g., replacing a RFID system by a GPS one). In addition, the component of this layer should format the data in an appropriate way to make it conform to the reference model and processable by the rest of the layers. Furthermore, it is at this level where we perform the correspondence between the sensed raw data and the entity owner of this data as

the provider know which entity it is supposed to supervise.

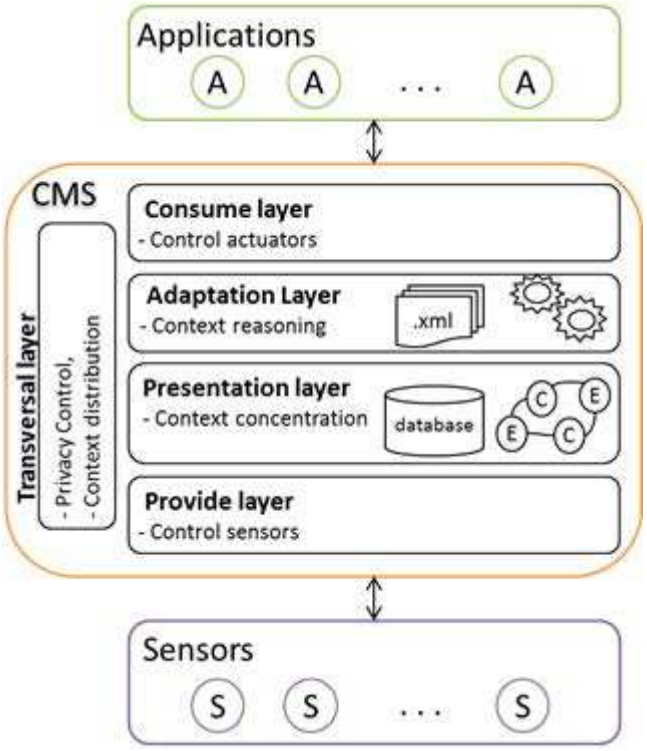

Figure 2.4: Layered framework for context-aware systems

The third layer (Presentation) (Storage and Management) organizes the gathered data about contextual information and entities to stored them in a database for persistence or maintain a representation of them in-memory. Not only sensed or deduced data have to be modeled, but also meta-data describing them (e.g., accuracy and recall, or life cycle information). In addition, it is responsible for creating a global view representing the situation of the different monitored entities, and maintaining the coherence of this representation. Furthermore, this layer should provide access to the managed data to components of the upper layers (e.g., 3rd party applications) in a synchronous or asynchronous way. In the first mode, the components use remote method calls for polling the server for changes. In the second mode, they subscribe to specific events of interest, and are notified when the event occurs (for example by a call back). The fourth layer (Adaptation layer) is responsible for reasoning and interpreting contextual information. It transforms the information returned by the underlying layer to a higher abstraction level (e.g., it transforms a GPS position to a position like at home or at work). In addition, this layer implements the generic process-

ing techniques which are frequently available in context-aware applications like mechanisms responsible for performing: filtering to eliminate un-appropriate information and triggering mechanisms to launch, under certain conditions, events that will be consumed by the receiving component. The fifth layer (Consume layer) is where the reactions to context changes are implemented (e.g., displaying text in a higher color contrast if illumination turns bad). Components of this layer are composed of third-party applications that subscribe to context changes to react appropriately. These applications have control over actuators to perform basic actions as a result of an adaptive behavior. The sixth layer (Application) is a collection of actuators specialized to perform a unique action. They react to the received requests from the context-aware application that wants to react to a context change by executing the corresponding action.

The transversal layer gathers the set of operations which are transversal and should be (or can be) applied on context information at each level of the different layers previously described. Examples of operations that can be found here include the distribution of context between the different components of the architecture, the control of access to context by a given component, interfacing the architecture at each layer with third party components.

2.4.2 Design Considerations

This section introduces the design considerations that should be taken into account to build an efficient context management system able to tackle challenges related to context-aware application developments.

Context delivery

Some context-aware applications need to continuously monitor the changing context of users and respond appropriately without much delay. As this kind of applications maintains always a fresh copy of the user context information, the adaptive actions that they may take are almost real time and highly reliable. But to maintain such an accurate vision, these applications need to receive every single update generated by the source of the tracked context information, thus they need to be able to handle a huge volume that may be received in short time interval. For example, a tracking system needs to be updated continually with measured position of the user or an object in order to perform an appropriate action like visualizing the exact position on a map in real time. A second kind of applications contains those that are unable to handle an important flood of information, and thus, they need a filtering layer that absorbs the important volume of updates generated by sensors and want to receive notifications for only event that satisfies particular predefined criteria. The definition of these

criteria is an important parameter for controlling the flood of received updates. A representative example of such applications can be a location-based messaging system that wants to receive a notification when the user enters or leaves a specific area to pop-up the post messages in this area.

Development support

The context management system should provide powerful tools for developers to mitigate the efforts required for developing and deploying context-aware applications. The main purpose of such tools is to hide complex operations which are commonly requested by context-aware applications like:

- Acquiring information from a sensor (e.g., GPS, temperature, noise, presence),
- Exchanging information between the different parts of the application that may be distributed across different devices or domains,
- Describing situation of interests by combining many conditions on context information of multiple entities,
- Creating adaptation rules by specifying trigger event, the condition that should be satisfied, as well as the corresponding actions that should be performed when the rule is verified,
- Invoking a service to perform a specific action (e.g., rejecting an incoming call),

Such tools can be distributed to developers in many possible ways; in the form of libraries, software development kit (SDK), or scaffolding, etc.

A library [60] comprises the implementation of a collection of behaviors for performing specific related tasks. It is used to ease software components reusability by hiding complex tasks and technical implementation details while providing an abstract access to developers. The wrapped behaviors can be invoked by external programs through a set of well-defined interfaces provided by the library via specific mechanisms. A software development kit (or devkit) [61] is typically delivered as a set of software development tools that support developers in the creation of applications. SDK empower software reusability and offers functionalities for building applications by assembling components. They are more general than libraries as they may be composed of many libraries each gathers an implementation of specific behaviors for a certain class of applications. Software companies may provide SDKs for free to encourage developers to use their system (e.g. Google with its Android SDK [62]), or provided through different licenses (e.g., GPL [63]) for evaluation or commercial use (e.g., Nuance with its Dragon Mobile SDK [64]).

Application scaffolding [65] is a code generation technique allowing to get started quickly with the development of a project. It gives developers the possi-

bility to create a simple application able to perform basic operations (e.g., manipulating objects). For instance, scaffolding is used by many Web development frameworks (e.g., Ruby on Rails [66]) to generate a web application that is able to manage (create, update, delete) entries from a database and corresponding URLs for a Web access. The code generation relies on a specification file writing by the developers to describe the application, and then the compiler uses this description to generate the corresponding application.

The libraries and devkits are language-depend, i.e. their implementation is written in a specific language, they can be used only by programs writing in the same language as them. In the contrary, scaffolding allows the description of the target application in a high level language and ignoring the implementation details as it is in charge of the compiler to map the description to the corresponding set of actions in the target language.

Deployment architecture

Due to the diversity of functionalities (e.g., sensory, actuation, storage, information sharing) implemented by a context-aware application, they are commonly distributed on different components of this application. This distribution enables the specialization of some components on specific tasks which facilitate their segregation and as a result the application maintainability. However, distributed applications are known to be complex to develop especially when it comes to deal with a variety of components which is the case for context-aware applications that have to retrieve information from a multitude of sensors. Another characteristic of context-aware applications is the abstraction level of the implemented functionality, an example of a low-level functionality is sensory which aims to provide contextual information to the rest of the application components. A context management system should provide support for distributing the application functionalities, regardless of their abstraction level (low, medium or high), across multiple components and provides a communication platform that hides the complexity of information exchange among these different components. Middleware [67] are the perfect fit for these requirements as they aims to facilitate software development without being integrated to them directly which also enable the reuse of the platform by other applications. Different architectures are possible for introducing such layer, for example centralized or decentralized architecture. In case the middleware layer is presented in a centralized architecture then a dedicated server is used to hold context information. This server provides interfaces to applications for requesting available context or subscribing for specific information to be notified upon they become available. It also hides context acquisition from end-application by providing another set of interfaces for sensors to publish the collected context. Thus, it enables the management

(i.e. development, deployment and maintenance) of sensors independently from the management of context-aware applications, which facilitate tremendously the developer task. The decentralized architecture is an alternative to the previous architecture that aims to avoid having a unique manager component that may become a bottleneck. The use of this kind of architecture implies the distribution of the context information on different places. The middleware layer takes in charge the components addressing functionality and the communication between them. A consequence of such architecture is that components of a context-aware application are not specialized in one function but may perform many ones, for instance sensing context and processing it. However, the additional features may make the components more complex and cannot be deployed on small constrained devices.

Programming abstraction

The development of context-aware applications is known to be a complex task for different reasons, for instance it is very difficult to understand the target execution environment in order to setup accordingly the application's adaptation rules. Therefore, it may be interesting to developers to have different programming levels that range from low (e.g., code in Java) to high level (i.e. abstract) and that can be used differently in the development process where it best fits. For example, for the complex and custom parts of the application it may be interesting to directly write code in an appropriate programming language like Java. While for less complex parts (e.g., defining the different parts of an adaptation rule) or parts that are susceptible for changes at the application run-time (e.g., trying different values for a given parameter), it may be interesting to use separate markup documents or configuration files and having access to them during other stages of the application lifecycle than the development phase.

The tools provided to developers through the context management system should provide support for high level programming. This should allow an abstract description of the components of the target application and if it is possible provide the ability to describe in a high level the inner parts of some components (e.g., properties, operations). In addition, it may allow for a high-level description for the possible interactions and exchanged information between the different components of the target context-aware application. From the developer perspective, the use of high level specifications facilitates tremendously the understanding of the application design view without carrying much about the target architecture.

The design view constructed with the specifications need to be translated into a working code that can be executed at run time or a skeleton that may be modified and customized by developers. At runtime the instantiated components may

become more closely coupled than at the design time with a potential distribution across different target environments. In addition, the elements of the markup document or method calls made by the components at design time needs to be mapped at runtime into requests to one or more services of the runtime framework or third-party services. An example of services provided by the framework includes a service responsible for routing messages between different components deployed on this runtime framework. The latter should be proposed as part of the context management system to provide the needed infrastructure for interpreting, instantiating and initializing the software components of the application and thus facilitating its deployment.

User empowerment

User empowerment [68] is a paradigm that relies on the assumption that users know exactly what they want to do. It aims to give to the users the adequate tools and a full control on them in order to accomplish a given task. As a result, the system becomes a passive element while the user is engaged and becomes an active element since the system has to receive direct commands from the user.

A system complying with this paradigm should provide different levels of control depending on the end-user expertise with regards to the provided controlling mechanisms. For instance, experts may have a complete and fine-grained control over different aspects of the system while less experimented users should have less rich control with a more intuitive interfacing. Furthermore, the system should be intuitive taking into consideration the human cognitive capabilities to ensure a low learning curve, as well as easiness of use and learn, for its controlling mechanism. In addition, it should tolerate users' potential mistakes by attempting to minimize them as possible and providing recovery mechanisms.

The empowerment of the different users (both developers and end-users), at their corresponding level of expertise, who are concerned by context-aware applications can address the inherent complexity of the development and usage of these applications. Complexity is a general aspect of software engineering that can be classified as proposed by F.P. Brooks [69] into accidental and essential complexity. The former is caused by the properties of software engineering tools chosen by the developer to perform his programming task. The latter is inherent to the characteristics of the problem that the developer is trying to solve, in our case it is the context-awareness.

The context management system is supposed to be interfaced with developers who are considered as experimented users. It should address the accidental complexity of developing context-aware applications by empowering developers through providing useful mechanisms for reducing the development complexity. The essential complexity is more related to the runtime phase and the adaptive

behavior that should be exhibited by the application. They can be addressed by empowering and involving end-users in the development of these applications but at a different stage than developers. In the case where complexity is due to the multiple choices available for the context-aware application when it comes to perform an adaptation, users may be asked to be involved in making the adequate choice and tailoring the application behavior with respect to their changing needs.

2.5 Summary

Traditional applications used to be developed in the lab where everything is under control and without worrying to much about the execution environment. However, developers implementing adaptive applications face the challenge of dealing with the dynamically changing user environment. This is a must in order to be able to move the application customization from the in-lab development phase to put it in the user hands during the operational phase. The dynamicity of the execution environment where the adaptive application will be running leads the application developer to define exactly which behavior that application should perform under which situation. However determining the exact set of situations to expect is quite challenging as it depends highly on the user and his/her environment, furthermore the corresponding behavior defined at priory by the developer may not match the user expectation and need at this situation.

The involvement of other actors than the developer or designer is important for such applications in order to correct and customize a behavior already defined for a given situation or define a new situation that was missed out by the developer and attach to it the accurate behavior that reflects the user perspective. The developer should develop efficient methods of adaptation, i.e. the intelligence behind the selection of the behavior to expose as a result to situation change that reacts. In addition, the developer has to develop the methods that will allow the users to customize the adaptation process. As a result, the adaptive behavior when executed will have more chance to match the user expectations.

The study of the state of the art on context-awareness computing shows, as presented earlier in section 2.2.1, that the notion of context is viewed from different perspectives and can be understood differently. As a result, involving other partners than the developer in the design and development of the adaptive application implies the need for the alignment of the perspectives and understanding of context to facilitate its communication across all these actors. In addition, these definitions promote the entity-centric viewpoint of context by separating entities from each other and representing the cloud of context information as a surrounding to these entities. Thus they are incomplete as they cannot repre-

sent connections between entities and describe these connections with contextual information.

Furthermore, the support for the creation of context-aware applications requires the definition of a comprehensive environment that empowers developers through facilitating the development and components reuse as well as empowering the end user by providing mechanism that helps them getting involved in the creation process.

In continuation to this chapter, the next one discusses in more details the general research problems and examines, at a more technical level, the state of the art proposals concerning the support tools facilitating the engineering of context-aware applications.

Chapter 3

Survey on existing approaches

The context-aware application perspective may not match the user perspective at 100%, which seriously threat to annoy the user with potential wrong actions if the application mainly relies on automatic adaptations without (or with little) user consultation. Consequently, context-aware applications are associated with a number of typical usability problems, and in some cases, a decrease of the system's usability outweighs the benefits of adaptation. Due to the lacking transparency of context-aware applications, the adaptation decisions are inaccessible to the end-user, which disallows an overriding of the behavior. Without control and means of customization the end-users will abandon useful context-aware services.

This chapter is the continuation of the precedent one as it surveys and provides an evaluation of the background works in the context-aware computing research area. However, this chapter deals with the technical aspects of the engineering of context-aware applications. It examines in details the existing frameworks and tools based on the support they provide for the development of context-aware applications. The present chapter is divided into three parts: section 3.1 establishes the research challenges facing the context-aware computing from the perspective of different stakeholders. It also derives a set of key requirements that represent a detailed extension of the general requirements presented in the precedent chapter. They will serve for the evaluation of state of the art works presented in this chapter. Section 3.2 will survey a selection of background works in the literature of context-aware computing. The summary section 3.3 evaluates these works based on the aforementioned requirements, discusses their limitations and it will motivate the research directions of this thesis that will serve the foundation for following chapters.

3.1 Research challenges

The current section enumerates the general research challenges that can be further investigated in the aim of facilitating the creation of applications enhanced with context-aware behaviors. They serve the basis for the evaluation criteria of the background research and the source from which the contribution of this thesis is derived. These challenges are segmented based on the actor (e.g. user, developer) concerned directly by the challenge.

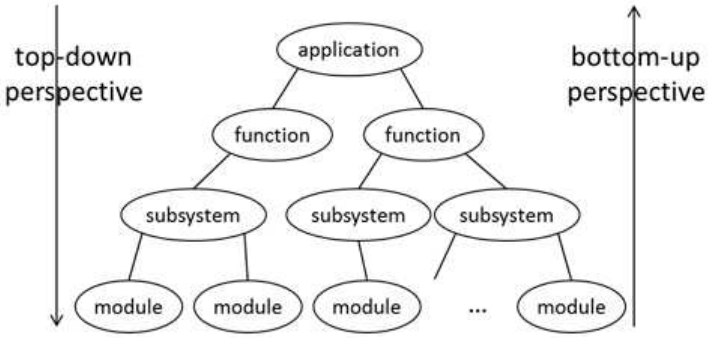

Figure 3.1: Application abstraction hierarchy

The conceptual diagram depicted in figure 3.1 illustrates the decomposition of an application into modules (e.g., mostly external modules like sensor/actuator), subsystem (i.e. component responsible for controlling modules), function (e.g., storage/distribution), and how the application can be seen at different levels of abstraction depending on the perspective. In fact, the design of a context-aware application starts by defining the goal of adaptation that will be achieved by this application. Then, the required context and user information are drawn based on the defined adaptations and the specific domain of the application. However, the development of the context-aware application takes the opposite direction and starts from bottom to up by first crafting the component responsible for getting data from sensors, then building the component responsible for storing the collection of data and the component that process context updates to make decisions, and finally the component which will perform the actions. Support both approaches is important but quite challenging.

In addition, given access to end users reduce the chance of the awareness mismatch [70] that rises when the user fail to understand the system adaptive behavior or when the system fails to meet the user expectation. Musumba et al. [70] illustrated the problems that arise from awareness mismatch through an example of the Car Navigation System. When this system relies only on current user location and the given destination to provide a route between them, the

decision by the system under these conditions is easily understandable by the user. However, if the system relies additionally on traffic conditions to propose a route then the decision of the system changes very often (i.e. not the same proposed route based on time of the decision). In this case, if the user has no knowledge of the use of traffic conditions for decision making (i.e. route proposal) then it will not be easy for him/her to understand and accept the adaptive behavior of the system. Achieving this goal is quite challenging and paradoxical as simplifying the context modeling to allow an easy perception of context by user end up making it hardly processable by the application. Moreover, to structure the context model in a way to enable an efficient processing make it not readable and not easily understandable by non-expert users.

3.1.1 Support for the data

Context-aware applications hold a perception of their environment in a knowledge base which mainly contains a context model as well as a user model. Part of the creation of the knowledge base by the developer is the shaping of these models. It is important to guarantee a standard understanding of how context is modeled in the knowledge base in order to enable other actors to access it and participate in filling it. As a result the transparency regarding the acquisition of context and its usage is crucial for reducing the gap between the user perceptual model of the system from one hand and the system's model of the world from the other hand. Furthermore, for the purpose of efficiency of context processing, a uniform representation and semantic of context should be shared among the different parts of the context-aware system as described in the previous chapter. The achievement of both goals is constrained by the consideration of the following aspects:

Context modeling

The context modeling approach is used to represent contextual information internally in a context-aware application and to be able to manipulate them easily. Beyond context representation, the modeling approach should also reduce the complexity of developing context-aware applications. Context-aware applications are most of the time distributed applications and have to share the context data among different components. Thus, the modeling approach should be defined carefully to support an efficient re-use and sharing of context. The approaches can be divided into two categories based on the strength of the underling semantic of the model. Models with strong semantic constraints imply for all components participating in the context management to understand the same representation and semantic for a given context data. This may ease the maintainability of

the application but limit its evolvability as only components sharing the same semantic can be integrated. The models with weak semantic constraints are more permissive as they don't force the compliance to a universal semantic and as a result make it easier the integration of third-party components. However, misinterpretation of represented information is susceptible to happen.

User modeling

In interactive systems, the system focus mainly on the user and thus the system relies heavily on an internal representation of the user and its characteristics. Context-aware applications need to evaluate the user characteristics, in addition to the user situation, to perform the corresponding adaptation that bests match the user expectation; as a result they need to integrate into their context model a representation of the user. Examples of such characteristics that can be used by many applications include his/her preferences (e.g. language, connection settings), interests (may serve as a filter for eliminated information of non-interest to the user), level of expertise in a specific domain (e.g., expert or novice). Additionally, physiological information (e.g. emotion, stress) may be important to represent for same applications however they might be harder to capture then other type of data. Representing the user internally in context-aware applications allows the maintenance of information on the user state which can further enrich the information on context and enable a more accurate context-aware behavior. Thus, it is crucial to provide support for enabling the integration of user-modeling along with context-modeling at the architectural level.

Context processing

The processing of raw contextual information can lead to a significant increase in the properties of the initial information to generate a knowledge which is more complex and expressive. Simple processing rules can be used to replace the initial information with another one of more ease to use for example replacing a numeric value representing temperature to a value from a finite state (e.g. red, yellow). Other more sophisticated techniques can be used for complex processing. Example of such techniques includes the fusion which consists of merging multiple readings of the same type of information into one value (e.g. the average function or the arithmetic mean). Another technique is the aggregation which consists of combining multiple instances of information of different types, which may have nothing in common, into one single composed structure (e.g. aggregating GPS location information, with civic address information in addition to a label provided by the user). Semantic interpretation is a transformation process that relies on a semantic model to map an initial input to something understandable

by the system, it is usually employed in speech recognition but can be also used for context interpretation. In addition, logical inference algorithms (e.g. forward-chaining, backward-chaining) can be used as a derivation technique.

Metadata handling

As introduced earlier in section 2.2.2 about context properties, the context cannot be reliably used without considering some meta-information that describes its quality. This meta-information should be integrated in the context model on which rely the adaptation process. In addition to the representation aspect, the architecture of the context-aware application should support the manipulation of the context quality by having operations that acquire it from context sources and supply it to the components responsible of the adaptation process. In this case, it is up to the adaptation process to make decisions on context data and include the supplied quality parameters. Furthermore, the process of manipulating context information through a one or a succession of transformation techniques, presented in the previous section, affect the initial properties of this context information. As a result, the context quality properties should also pass through a transformation process to be synchronized with the context information it describes at each level.

3.1.2 Support for the design

The design support should provide a complete view of the context-aware application. It should cover the end-to-end functional decomposition of the application representing the different levels of context management (i.e. acquisition, abstraction, reasoning, and actuation). In addition, it should support the visualization of the flow of information through the functional components of the application established as a result of the context processing pipeline. Furthermore, it should enable the participation of different actors in the process.

Design process

A design a procedure that relies on the acquisition method for choosing what context to process is not suitable for context-aware application and may limit their kinds. Applications developed in controlled environments (e.g., in lab) tend to lead the focus of research on the implementation aspects and neglecting the design process for instance the design might be described informally with detailing any process guidelines or description of main design issues which are important elements of engineering of any software. A design process specifies the context requirements for an application and it is assisted by tools for creation, deployment and administration of context-aware applications.

Information flow

The establishment of a standard architecture gathering the different components of the system (e.g., computation, storage elements) along with well-defined interfaces between these components enables a seamless understanding of the static aspect of the system which concerns the components role and structure. Additionally, to understand the dynamic aspect of the system an inspection of the invocation process of the system's components is needed. The combination of both aspects (i.e. architecture structure and process pipeline) forms the foundation of the information flow within the system. This flow is characterized by handling two different types of information context information about the application environment and information about the internal components of the system. For conceiving architectures with modular and reusable components with well-structured information processing, it is important to draw the flow of information between the system building blocks including type of information and flow direction.

Expertise levels

It is not easy for other actors than developers (e.g., designers, integrators) to take part of the creation process of context-aware applications as they don't have the required skills to deal with such complex systems that have heterogeneous components and devices. Consequently, the support to the creation of context-aware applications should be tailored for these actors' expertise and development skills, in addition to the basic development support.

3.1.3 Support for the development

The logic in context-aware applications is most of the time directly implemented in the system behavior and usually tightly coupled to it leading to a rigid implementation which is difficult to maintain. Developers tend to focus on some tasks of the implementation of context-aware applications like how to acquire context and how to persist it in backend servers; while neglecting an important component like the one responsible for the adaptive behavior and which needs the more important part of the development effort.

Holistic view

In many research works in context-aware computing, an important amount of the energy is focused in the components responsible for the acquisition of context with less focus on other actions like the control of elements responsible

for performing actions on the real world, performing actions that modify the internal state of the application, or measuring the impact of these actions.

Changes in context will modify the information value in the context model or even the structure of the later. When a certain condition is met by the information in the context model, a specific component of the context-aware application should select what behavior the application will perform. Quality information if available can be used by this component to assess its confidence regarding this decision, for instance it may simply ignore the condition or trigger a mechanism that will stop acquiring context from the sensor providing low quality. Another component may decide on how to map the selected behavior to an appropriate action and which actuator to invoke. After executing the action, feedback from the application should be collected about the status of the action (e.g., terminated with success or failure, pending, etc.).

Consequently, the support to such application development has to provide a fine grained view of each component of the application and make them reusable across applications or across components of one application. While the design support should abstract the different parts of the application (e.g., by representing them as black boxes with input and output interfaces), provide a holistic view of the application and make the building blocks (e.g., adaptive behavior) reusable for a wide range of applications.

Context-awareness Integration

Providing context-awareness to a legacy application that does not have this feature can be performed at different levels: a low level support would be providing simple functions like the access to sensor readings, a higher support level would be providing tools and infrastructure that facilitate the application design, development and deployment. The ability to integrate, at multiple levels, the context-awareness to an application is constrained by the availability of easy-to-use interfaces provided by the infrastructure as well as the modularity of the application itself. It is apt to the developer to choose which support level is needed for his application by distributing the different parts of the application between the components already implemented in the application and the components of the support infrastructure enabling context-awareness. In some cases, such distribution is not enough and a deeper integration is needed between the application and the context-aware behavior; for example in case of the adaptive behavior tend to deliver different content based on context data. This integration leads to a closely coupled context model to the data model contained in the application.

Programming paradigms

Context-aware application design principles need to be specified into programming paradigms. The formalization of these paradigms provides developers a set of what help making the final application comply with a standard architecture style and reduce the application development overhead. The aspect that could be concerned with these paradigms includes:

- Context acquisition: providing a consistent methodology to deal with sensed data of any type (e.g., implicitly acquired via sensors, or explicitly via forms), abstract the interaction with the source (i.e. the sensor or user) and unify the representation of the acquired data.
- Context processing: provide abstraction for the different possible context processing techniques as the acquired information may serve for context management operation or for reasoning on automatic actions to perform. For instance, supporting mechanisms for detecting situations, or events triggering. Additionally, it may facilitate the control of adaptive behavior while the application is underdevelopment or operational. The support may reach other specific needs for some applications like establishing relations between context and content on the fly and the use of context for filtering content, etc.
- The integration of context-awareness with user modeling might be facilitated by the supported of personalization and the integration of some generic techniques into the development support enabling user modeling.

Application re-configurability

It's often that the context-aware behavior is developed as an internal part of the application this is mostly the result of (potentially wrong) assumptions made during the application realization by the developer about the operational environment. The context-aware applications may be operating in different environments with the presence of other systems which are potentially heterogeneous. Under such conditions, it is important to provide mechanisms enabling the configuration of the application to avoid errors in the application behavior, for performance tuning. Providing developers with such underlying mechanism will facilitate the parameterizing the sensors and actuators, for example turning off the components providing poor quality.

3.1.4 Support for the use

It is common that end-users are not involved in the design and development of context-aware applications and little information are given to the user regarding the context-awareness of this application. Such behavior could potentially lead to

usability problems [70] and requires the consideration of extending the actors to others than just the developer at the different levels of application life-cycle management (i.e. design, development, deployment, administration) and especially in the adaptation process.

Involving end-users

Making automatic behavior as response to situation detection tend to be error prone and limit the user to a passive role, in the other hand, enabling the user to perform actions that will tailor the system behavior tend to overload the user and may require him to conduct complex actions. Both approaches should be integrated to enable the system ask the user for help to resolve ambiguous or provide answers for incomplete information, and it also enables the user to delegate some complex and recurrent tasks to the system for automation. The approach for this integration requires providing different level of control to the user over the application depending on its expertise in order to configure the behavior of the system to his/her needs. The reconfiguration mechanism should not imply the re-implementation of the application.

Architecture transparency

In addition to providing control, it may be important to help the user understand the operational aspect of the application to fine-tune it for his/her needs. For this the application has to expose its structure (for example by providing for each one of its components a human-readable description), internal state (for example whether a sensor/actuator is on or off) as well as configuration information that will parameterize the launch and operations of the system. Further transparency may be needed regarding the decisions and operations of a context-aware application especially when the impact is of importance to the user. For example, for the context acquisition or interpretation a feedback should be provided to the user for cases when there are errors and imprecisions.

User privacy

Context-aware applications collect a considerable amount of context data that represent very sensitive information about real users (e.g. personal/location information). The use (e.g. collection, distribution) of such data must be controlled by the owner user as most of these data concerns information that people may not like to share with anyone. Furthermore, the user should be made aware about each in progress action that concerns his contextual information, for example he may be notified about what exactly has been collected and may be given the ability to allow or forbid this action. The architectural support need to formalize

how the operations made on context can be controlled at each level of the different components of the application. Furthermore, it may be interesting to divide the set of controlled sensors into different classes (e.g. based on sensitivity of the sensed data to high/medium/low) in order to provide a fine-grained control.

3.2 Related works

The current section studies the literature works related to frameworks and infrastructure enabling the rapid development of context-aware applications. These works are examined in order to highlight the underlying architecture of the approach and how it addresses the previously defined research directions. The different works are classified in two classes general and specific purpose platforms based on whether the proposed solution aims to target a specific kind of applications or can be used in a wider range of domain applications.

3.2.1 Specific purpose platforms

This class gathers the literature works that provide a support for the engineering of a specific kind of application, for instance location-based applications.

PLASH

In [71], the authors proposed a platform for supporting the deployment of Location-based service (LBS), and the contribution of users who provide location-related data. The users contribution may solve difficult location awareness problems like real-time surface traffic estimation, city panoramas, and social networking analysis.

Figure 3.2 illustrates the layered architecture of PLASH, the Communication layer support different wireless communication technologies (3G, WiMAX, WLAN) and networking contexts vehicule-to-infrastructure (V2I), vehicule-to-vehicule (V2V)). This layer provides an API for applications to specify the communication requirements (e.g. bandwidth), it also provides an Representational State Transfer (REST)/XML API to support different devices, and RESTful (Representational State Transfert) as a common protocol for the different servers (e.g., application and database servers).

The Data layer is responsible for representing and storing data, managing queries. A geo-location database is used to store location data. At this layer, a geo-query processor supports different kinds of Location-dependent Spatial Query (LDSQ) like select, range, moving objects, continuous, and k-NN queries. A Social Core database is used to store users' information (e.g., username, password,

and friends). For privacy, location data are filtered before been processed (e.g., dropping username from location information).

The Service layer provides fundamental services like authorization control services (e.g., login/logout, register and friendship), database access services (e.g., select, insert), and LDSQ services. It also accommodates mature LAS applications.

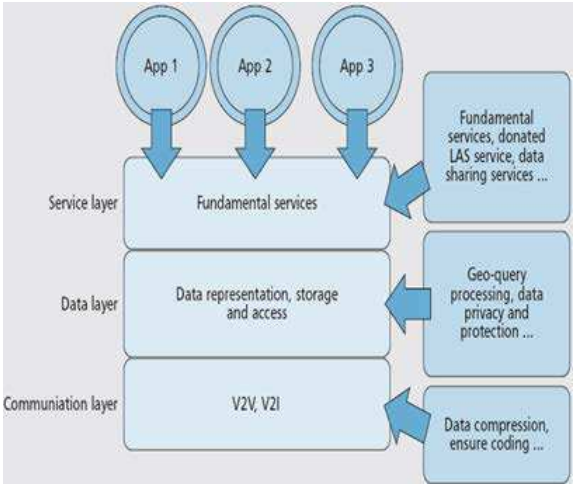

Figure 3.2: PLASH platform

The interaction between these different layers is illustrated in figure 3.3 which represents the operating model of PLASH that is divided into a PLASH application and a PLASH platform. PLASH application contains a presentation layer, and a logic layer. In the presentation layer, software clients are responsible for the user-side presentations and interactions. The logic layer implements application logic, which can be hosted by the PLASH machine or run on its own machine. Different communication protocols are used between layers: HTTP/RESTful between the presentation layer, logic layer, and access managers in the service layer; Java Message Service (JMS) within different components in the service layer; Java Database Connectivity (JDBC) to access PLASH's databases.

In the PLASH platform a set of procedures enable applications to access/share services and databases. The Personal Information procedure (steps 1 to 7) allows login/logout and maintaining user personal information (e.g. friendship/membership, username/password). Upload procedure (steps 8 to 11) is initiated once the user has successfully logged in order to send application dependent information (e.g. location information, application ID/API key, username) to the PLASH platform. Geo-query procedure (step 12) may be activated to check the satisfaction of a geo-query. The Feedback procedure (steps 13 to 15) is used to return in-

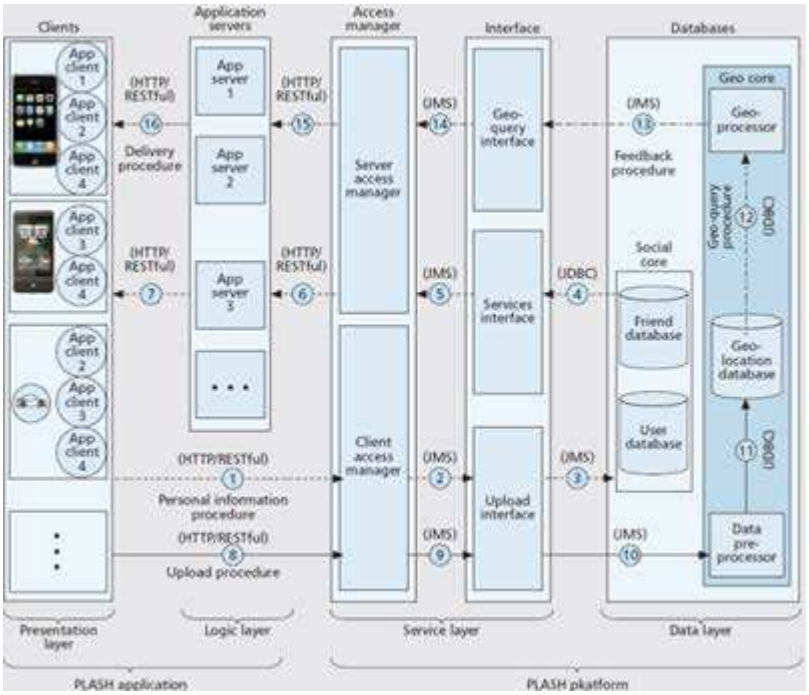

Figure 3.3: PLASH operating model

formation to the application server after triggering the application's associated services. Then, the application server processes the returned information based on its application logic and delivers the aggregated data to the mobile device (Delivery procedure, step 16) that will display the result.

The platform provides an Application Specification Language (ASL) to enable application builders creating context-aware applications. Builders must request an API key to their application; the key will be used to control access to the PLASH's interfaces. For the creation of the application server, an ASL file must be uploaded to the PLASH platform. This file contains three parts: Service Usage Description (SUD), Application Description (AD), and Database Definition (DD). The SUD defines the set of services (e.g. geo-query, personal information query) to be used by the current service. The AD defines the schema of the private database to be used by the service. The AD provides general information about the service as well as the definition of the services to be shared. The main goal of PLASH is to support developers in rapidly building context-aware applications with a focus on location as the primary contextual information. It does not involve other actors in the creation process; neither has it allowed the user to understand the adaptive behavior of an application deployed on the platform. However, users input is considered in some usage scenario for instance in case of lack of information or

ambiguity. In addition, user information (e.g. interest) are integrated into the context modeling as illustrated in a targeted advertising scenario of a shopping application but no support for quality information is provided.

Omnipresent

Omnipresent [72] is a context-aware platform based on the service-oriented architecture. The default services within this architecture are a LBS (via a map presentation, routing, advertisement) and a reminder service. Omnipresent employs push services (i.e. services that users don't have control on their invocation) and pull services (i.e. services requested explicitly by users). Example of the first class of services is the delivery of maps enhanced with Point of interest (POI) information. The aim of these services is to contextually help users finding products, services, POIs, friends and management of their daily tasks. Omnipresent services are supposed to be installed on the server side where is implemented the whole intelligence.

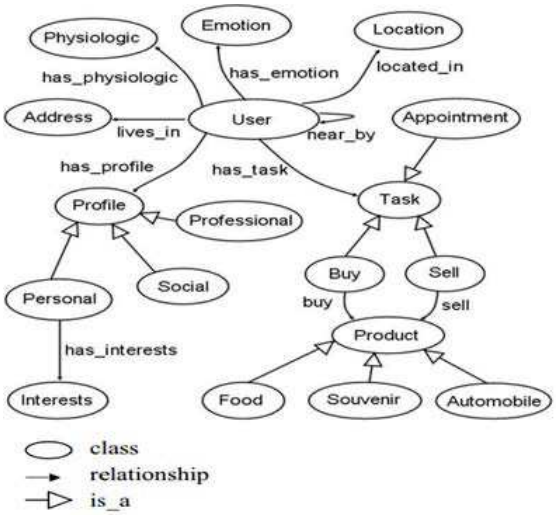

Figure 3.4: User context ontology

The context-model 3.4 of Omnipresent is based on ontology expressed in OWL, and the reasoning on rules expressed by the use of Jena framework. The different classes of the ontology are:

- User class is the main concept;
- Profile class is divided into context categories: Personal profile (e.g., user interests, marital status, date of birth), Professional profile (information about job activities like title, email, company), Social profile (information about social behavior like religion, smoking and drinking habits);

– Task class contains interests on buying or selling products that may be advertised (push services) based on the user context (e.g., the user is located near to a shop. This class contains also scheduled meetings which are represented by the Appointment class. Users are reminded for their appointment based on time, place and user context. Examples of conditions under which a reminder to make a call to a friend is triggered are the emotion status of a friend is happy and the caller and callee are idle;
– Physiological class contains user physiological state (e.g., heartbeat, temperature, blood pressure);
– Emotion class contains emotional status (e.g., sad, happy, nervous, badhoumour).

To enable reasoning, users may specify rules for triggering actions whenever the context changes.

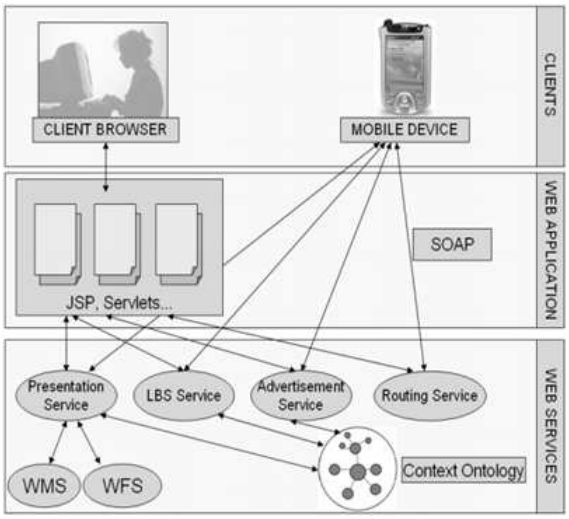

Figure 3.5: Architectural Design

Figure 3.5 presents the different services of Omnipresent: LBS web service, Presentation web service, Routing web service, and an Advertisement web service. Omnipresent provide also a client application for users to state explicitly some of their profile information, appointments, and daily activities. The different services are as follow:

– LBS Web Service responsible for receiving and managing user context. It provides operations like user registration, updating the user location, its emotional and physiological status, and its appointments;
– Appointment Web Service responsible of managing users' products and services. This service extends the Directory Service of the OpenLS Service

Specification with the use of XMLSchema to describe products and services.
- Presentation service responsible for providing maps of a geographical area. It extends the OpenGeospatial OpenLS specification with the ability to analyze user profile for providing maps augmented with POI. This service works with two others: WMF (Web Map Service) from which the presentation service obtains the maps, and MFS (Web Feature Service). The MFS enables the user to query and update geospatial data in an interoperable way.
- Routing Service responsible for providing on demand routes among different places. The user can specify on his request some preferences to be taken into account for calculating the routes, e.g. traffic conditions. Roads and intersections are stored as a graph partitioned into non-interleaving subgraphs. Paths in the graph are pre-calculated and nodes may belong to more than one partition.

Omnipresent combines a limited types of information which are either sensed automatically or acquired from the user. It relies on a proprietary ontology to represent context as well as user information to ensure a unified understand and unique sematic of the represented information along the different components deployed on the infrastructure. As the applications are server-side, they are shared by the different users who are not allowed to change/customize their behavior or they have visibility on the adaptation process. The adaptation process is not flexible as the triggering conditions are based on ontology classes that should beforehand be defined, i.e. to add new type of conditions a modification of the core ontology is required.

3.2.2 General purpose platforms

This class gathers the solutions that provide a support for the engineering of a wide range of applications, for instance communication or collaboration applications.

SOCAM

In [57], the authors started by defining an OWL-based context model, then they proposed an OSGi-based infrastructure (SOCAM, Service-Oriented Context-Aware Middleware) for building context-aware applications in smart-home environment. In SOCAM, context is represented in first-order predicate calculus as Predicate (subject, value), where subject can be a person or location or object, a predicate can be “located in” or “has status”, and value can be “living room”, “open”, “close” or “empty”. This basic model can be extended by combining the

predicate and Boolean algebra (union, intersection, etc.). For example: FoodPreference (familyMembers, foodItems) → FoodPreference (John, FoodList_1) v FoodPreference (Alice, FoodList_2) v FoodPreference (Tom, FoodList_3) The ontology describing the context model is a collection of RDF triples in the form of (subject, predicate, object), where predicate is a property relationship, subject and object are the ontology's object.

The context ontologies (next figure) are dividing into a high-level ontology (to capture general information about the physical world) and a domain-specific ontology (to define details of general concepts and their properties in each subdomain like home or office). This division helps decreasing the complexity of managing heterogeneous context data.

In the next figure (3.6), an instance of the ContextEntity class exists for each distinct user, agent, or service, and has a set of descendant classes: Person, Location, CompEntity, and Activity. These basic concepts are detailed by the domain-specific ontology.

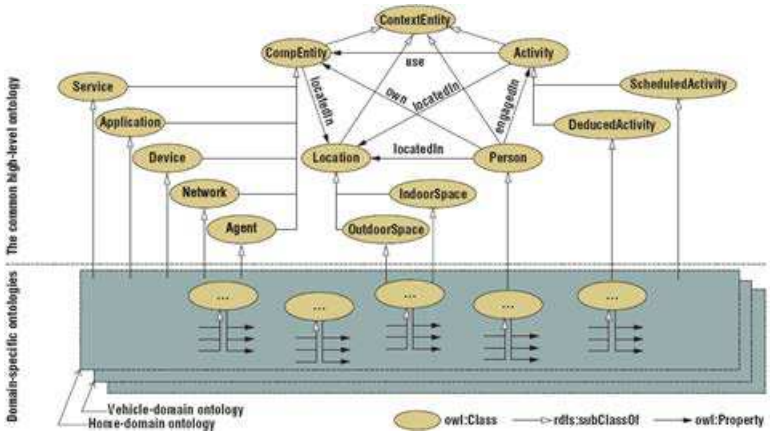

Figure 3.6: Class hierarchy diagram for SOCAM context ontologies

Figure 3.7 presents the architecture of SOCAM, which is composed of:

- Context Providers retrieve sensed context from different sources (internal or external) and convert it to an OWL representation to be easily shared by others SOCAM components. An API is designed to support both context query and context event subscription;
- Context Interpreter responsible for logic reasoning. It is composed of a context reasoner that deduces high-level context and resolve context conflicts, and a context knowledge base that provides an API to query, add, delete and modify context knowledge;
- Context Database stores separately context ontologies and past contexts

for each domain;
- Context-aware applications use different level of context to adapt their behavior accordingly. These application are designed by specifying Actions triggered after the occurrence of a certain Event (e.g. context changes) and with respect to a set of Conditions (e.g. occurrence of previous events);
- Service-locating Service (SLS) allows Context Providers and Interpreter to announce their presence in order to be located by users and applications.

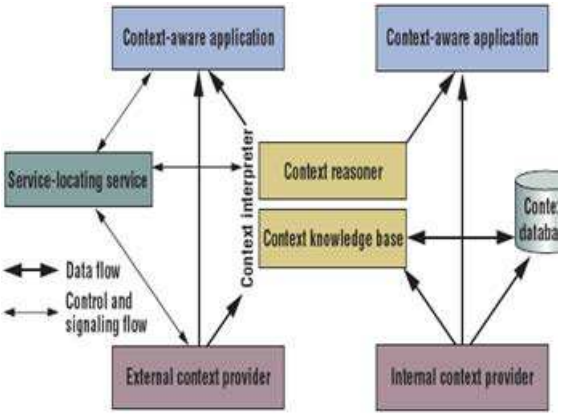

Figure 3.7: Overview of the SOCAM architecture

The ontology's reasoning mechanism, in SOCAM, supports RDF schema to perform RDF schema reasoning, and OWL Lite to describe properties and classes; relationship between classes (e.g. disjointness); cardinality (e.g. exactly one); equality; characteristics of properties (e.g. symmetry); and enumerated classes.

The system infrastructure consists of (figure 3.8):
- OSGi-compliant residential gateway manages communication from and to various local networks. It can be used to attach networked devices, electronics appliances and sensors via different home network technologies. It also host context-aware services downloaded from Context-aware service providers;
- Gateway operator manages residential gateways and their services;
- Context-aware service Providers offer to home users a set of context-aware services which can be combined into bundles to be downloaded on the gateway.

Figure 3.9 presents the OSGi software stack installed on the OSGi-compliant residential gateway. It hosts basic services like user and configuration management, permission administration. SOCAM components are developed above this platform as independent bundles using the provided APIs. These APIs require the caller to have necessary permissions; also services need certain permission to

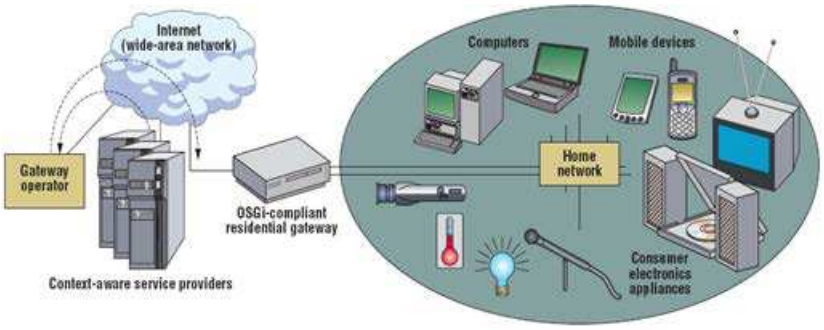

Figure 3.8: The OSGi-based context-aware infrastructure

perform their tasks. Each OSGi bundle is associated with a Permission class. To verify an instance of this class, the SecurityManger must be called. It is the gateway operator that configures the gateway's system policy to grant the required permissions to bundles installed and running on it.

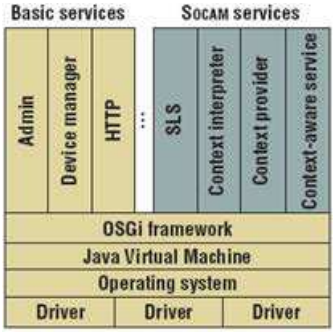

Figure 3.9: Software architecture of an OSGi-compliant residential gateway

An example of a context-aware application developed on top of the SOCAM platform is the dining room application. It uses an indoor location provider to locate people and a context interpreter to derive room activities (e.g. dinner). The context-aware behavior is specified by a set of methods invocated for a certain context. For example, the application plays music and adjusts light when family member have dinner. SOCAM relies on an extensible ontology-based context modeling and provides predicate-based reasoning approach for the generation of new information. User information are not integrated though they can be defined separately by extending the Person class. SOCAM provides support for the acquisition, reasoning and integration of contextual information, also it support the reusability of components (e.g. providers) thanks to its OSGi-based architec-

ture. However, it does not provide any specific mechanisms allowing the user to be involved in the definition of the adaptive behavior or the control of how his information are used by (or circulate between) the internal components deployed on the infrastructure or external ones.

Vimoware

In [73], the authors present Vimoware as a toolkit that support the development of web services on mobile devices for collaborative purposes. Figure 3.10 illustrates the different components, which are implemented as web services, of Vimoware:

- The Lightweight Web Services Middleware hosts and advertises Simple Object Access Protocol (SOAP)-based web services which are created by extending an abstract Java class;
- Artifact Management shares multimedia documents through HTTP;
- Device and Location Sensors provide gathers information about devices (CPU, network, memory, and device profile) and current status of the device. The information can be accessed by a client application directly via XPath/XQuery requests or by subscribing for any information. Implementing components as Web services enhance the interoperability and integration of different context information sources.
- User and Team Management for provisioning of user and team profiles. The profile are in XML format and gathers basic information (name, ID, skills) to associate human with services on the device. This association can then be used to search for a specific service provided by a person;
- The Flow Execution Engine (FEE) supports the execution of flows of tasks defined by an XML schema. The used language to describe flows allows a task receiver to reject a task if he is authorized, to assign a task to more than one team member, and to be able to execute a task in synchronous or asynchronous model.
- The Communication component provides instant messaging features.
- The Task Handler controls the execution of tasks (sent by the FEE component) in a giving device.
- The Flow Editor supports users to design their own workflows to be executed on a FEE.

The advertisement and discovery of services are implemented as a P2P-based subscription/notification approach to allow an immediate notification and reducing network traffic, which is suitable for ad hoc environments. The different Vimoware instances advertise themselves via UDP multicasting. Vimoware supports the hosting and advertising of SOAP-based web services, sharing of multimedia documents through HTTP, the provisioning of contextual information

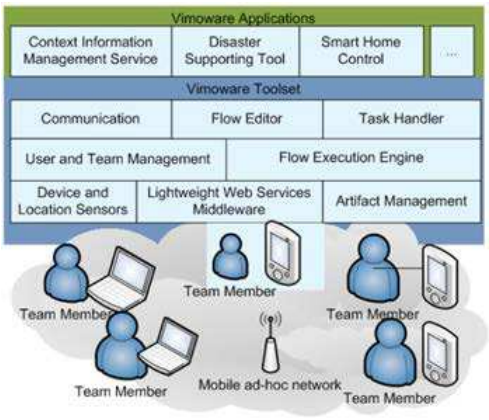

Figure 3.10: Components of the Vimoware architecture

about devices (e.g. network, CPU, memory) and users (e.g. name, skills), management of users and teams, the management of flows of tasks (creation, control and execution), the support of communication (e.g. Instant messaging (IM)). The middleware can be installed on different devices; it is able to make them communicate but doesn't provide an explicit management of historical data. Vimoware empowers end-users in the adaptation process throw the Flow editor that allows them defining personal tasks but does not provide mechanisms for controlling access to user information by services deployment on the middleware.

3.3 Conclusion

3.3.1 Discussion

Research works in context-aware computing aimed mainly to provide infrastructure facility for operations related to context management like collection, modeling and storage. Two distinctive architectural approaches [74] have been used to design context-aware applications: architectures that follow a broker model and those based on a point-to-point model. Both models contain two types of elements: context providers in charge of collecting contextual data, and context consumers (i.e. context-aware applications) that use contexts to adapt their behavior. In the first model, a context broker is used as an additional element to decouple context providers and consumers, limiting or eliminating the direct connections between them. This broker is thus, in most cases, in charge of context modeling and inference [74] [38]. However, these tasks are performed in a pre-defined way and they are barely customizable by context consumers.

As a result, the generated information may not fully match the specific needs of consumers. In the second model, context consumers know the providers and send their requests to them directly [75]. This model is less sophisticated; context consumers need to know which provider should be addressed for any given contextual information and should also be aware of their state (e.g. awake or asleep).

In both cases, consumers continuously request contextual information from their sources (either the context broker or directly from context providers) or subscribe to be notified with context updates. Upon reception, this information must then be processed by the context consumers to determine if it would impact their behavior. When the updating load increases, consumers become overloaded with messages, many of which are not at all relevant to them. Moreover, consumers have to store and handle context information locally to maintain a consistent vision of the user's situation and to adapt their behavior accordingly.

Regarding the context representation, earlier works [7] tended to use a simplified modeling approach with weak underlying semantic, lacking of expressiveness and limiting the reusability of the model. The research works then gradually moved toward the use of more modular and complex modeling approaches [1] that incorporated a lot of context properties as well as the relations amount these one and empowered the reusability of the model. Moreover, some works [57] relying on ontological approaches for the modeling enabled the integration of the raw context acquired directly by sensors along with semantically enriched context generated through a reasoning process. This can be considered as an important step to the integration of context modeling with the context-awareness behavior. Nevertheless, most of the different examined works do not consider the integration of the user model with the context model and most of them do not particularly take into account the user modeling or the extraction of user characteristics from the observed context.

Regarding the engineering support of context-aware applications, the different research works focus on the support of the implementation of the applications without considering the other phases like the design as well as the involvement of the other actors than the developer that may play an important role during these phases. Regarding the development support through programming abstractions that aim to reduce the development complexity, the examined works weakly address this question or propose basic support through abstraction or persistence for instance. It may be necessary to try new approaches for programming support that relies on the use and reasoning on highly abstracted context.

Regarding the context quality, most of the reviewed works agree on the importance of the consideration of quality information that describe properties of the source of context. However, not all of these works propose support of the association of quality information along with context information or furthermore

propose their integration in the context-aware behavior. McKeever et al. [76] proposed a modeling approach for representing quality information as well as their integration in the process of knowledge generation from the raw context to semantically rich information.

Regarding the openness of the applications to the user and the involvement of the latter in the context-aware behavior in the aim of avoiding usability issues, few research works addressed this challenge. Some works [28] limit the involvement of users to the definition of preferences information or policies in order to suitably customize the application to the user. Others [27] tried to incorporate users in the definition of the context-aware behavior of the application but they required from the user a considerable programming expertise.

3.3.2 Summary

This chapter reviews the literature works in context-aware computing from different perspectives. It firsts introduce a set of requirements that goes beyond the simple development support. Then, it surveys the related works and discusses them according to these requirements. This thesis emphasis the challenge of offering a flexible support for developing context-aware applications without constraining this support as a result of the underlying technical implementation. Although most of the reviewed works provide a significant support for engineering context-aware applications, they lack flexibility as a result of enforcing the one-size-fits-all model. In fact, these solutions usually propose a unique semantic for the managed context data and a uniform way to process these data for any kind of applications. The following chapters present the conceptual and detailed technical aspects of the thesis contribution, as well as they discuss the contribution validation through case studies implementation.

Chapter 4

A new framework

The survey of the state of the art efforts in context-aware computing allowed us to understand the core functionalities and operations of a typical context-aware application. In addition, reviewing the literature proposals for addressing the research problems in terms of frameworks and development support tools showed a variety of limitations. For instance, some tools lack flexibility as they were designed for a specific kind of applications; other works force the applications to share a proprietary context model (and as a result the corresponding semantics) with the proposed platform, etc. Therefore, the thesis through this chapter is concerned with the proposal of a more flexible context modeling approach as well as on a customizable reasoning procedure, then exposing these functionalities to developers in a suitable manner to support the engineering and maintainability of context-aware applications.

Our contribution aims mainly to tackle the complexity of developing context-aware applications by reducing the dependency between the different operations of such applications. This chapter introduces a context management system that hosts some of the context-related operations and exposes an interface for developers to program these operations. It uses a flexible context representation based on XML allowing the composition of context information and enabling applications to request the exact piece of information they need. It also enables the reuse of software components, for instance the information published by a given sensor can be used by different applications. In addition, this chapter introduces a new context representation that tackles the limitations of already established ones by reconsidering the literate viewpoint toward context data. The proposed representation aims to provide a natural way to describe relations between entities and as a result simplify the understanding of context for the different actors (and especially for users). In addition, it attempts to simplify the modeling of the knowledge contained in context-aware applications on both context and user. Moreover, it approaches the operational aspect related to

contextual information by proposing foundations for their usage in context-aware applications and emphasizing the dynamicity of contextual information.

The remaining of this chapter provides conceptual details of the proposed context management platform and explains at different levels the corresponding architecture. Then, it presents the underlying context modeling which is based on a graph representation. In addition, it explains the mechanisms provided by the platform for the specification of context processing as well as the different modes of reasoning and processing provided by the platform. Also, the chapter presents the support provided for preserving users' privacy. Finally, a summary concludes this chapter.

4.1 Holistic Architecture

Developers need to be provided with a comprehensive development environment that will support them in building context-aware applications, deploying them to the target environment and then using the captured context at runtime to achieve the target behavior. The realization of such a development environment requires the definition of a comprehensive software architecture that guides the engineering process of context-aware applications. This section describes in details the concepts behind the software architecture that correspond to the core contribution of this thesis. The main advantages of this proposal regarding related works is that this software architecture covers all relevant aspects of application development, and is universally applicable to a variety of application domains so that it can support the development of different kind of context-aware applications.

4.1.1 Conceptual view

From a high level of the implementation viewpoint, the proposed context management system is depicted in figure 4.1. Its architecture is based on the consumer/producer design pattern, with the introduction of a distributed broker. The Context Provider (CxP) represent the source of context, while the Context Consumer (CxC) represent the applications that consume context to adapt their behavior. The Context Broker (CxB) is used to decouple consumers (who subscribe for desired contexts) from producers (who publish context information). This broker is distributed between the device and the platform sides: CxC and CxP are seamlessly on the device or on the service platforms. The platform-side broker has knowledge of the context information from all providers and from the one produced after processing historical data. While, the device-side broker has knowledge of local context, it may request more global information from the for-

mer broker if needed. Moreover, a context history module stores all the context information published by every CxPs, whatever his location (device or platform side). This enables to log an historical view of the context in order to process it with data mining algorithms and then to use it in a deferred way.

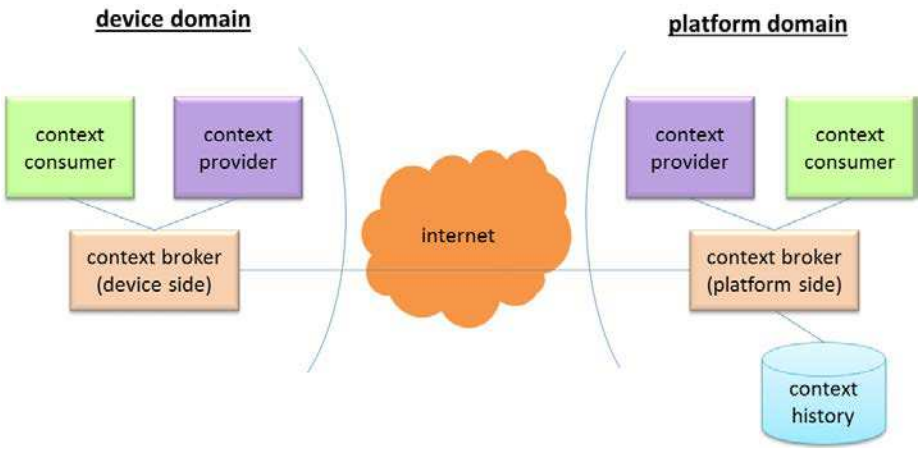

Figure 4.1: Distributed architecture of the Context Management System

Communication between the various components of the framework is either based on synchronous (request
response) or asynchronous (publish
subscribe) communications. In the asynchronous mode, the consumer subscribes to a type of context information and is then notified by the broker when a provider publishes such information. In the synchronous mode, the consumer requests context information from the broker that responds immediately, as it is continually gathering information from all the providers in the context history module. The asynchronous mode is likely used for instant consumption of context information, while the synchronous mode is used for deferred consumption. Through the symmetry of the framework's components implemented at the device and the platform side, it enables the adaptation on both device and platform sides (our second segmentation axis).

4.1.2 Functional view

At a more detailed level, the functional view of the context management system is depicted in figure 4.2. The figure summarizes the functions performed by the main components of the system and it illustrates the interaction between these components. The CxPs are responsible for sending context data to the Context Management Platform (CMP) which is in charge of managing context and its

distribution. The CxB is responsible for distributing context from sources (CxPs) to its consumers (CxC). A local context broker is used at the device level as a Cache to store context data locally (and temporarily until expiration) to speed up responses to future context requests. CxPs installed on user devices should publish their produced context to the local CxB that is in charge of forwarding it to other components. A Configuration Manager (CM) manages the configuration files used for custom processing of context data and assures their validation. A context repository is used for context persistence; in addition a rules repository is used to store the context reasoning rules. A Reasoning Engine (RE) is responsible for applying the reasoning rules defined by the context-aware applications to the context events generated by updates from CxPs. The Notification Manager (NM) represents the endpoint exposed to the consumers in order to provide subscription management functionalities, i.e. allowing the consumer to subscribe to certain events to be notified upon they occur. A Permission Manager (PM) is used to enforce user policies, such as which components (CxC) are allowed to request a user's contextual information. A lookup table (LT) is used to hold a correspondence between the context address and the components responsible for publishing updates on this address as well as the components subscribed to be notified upon the context is updated.

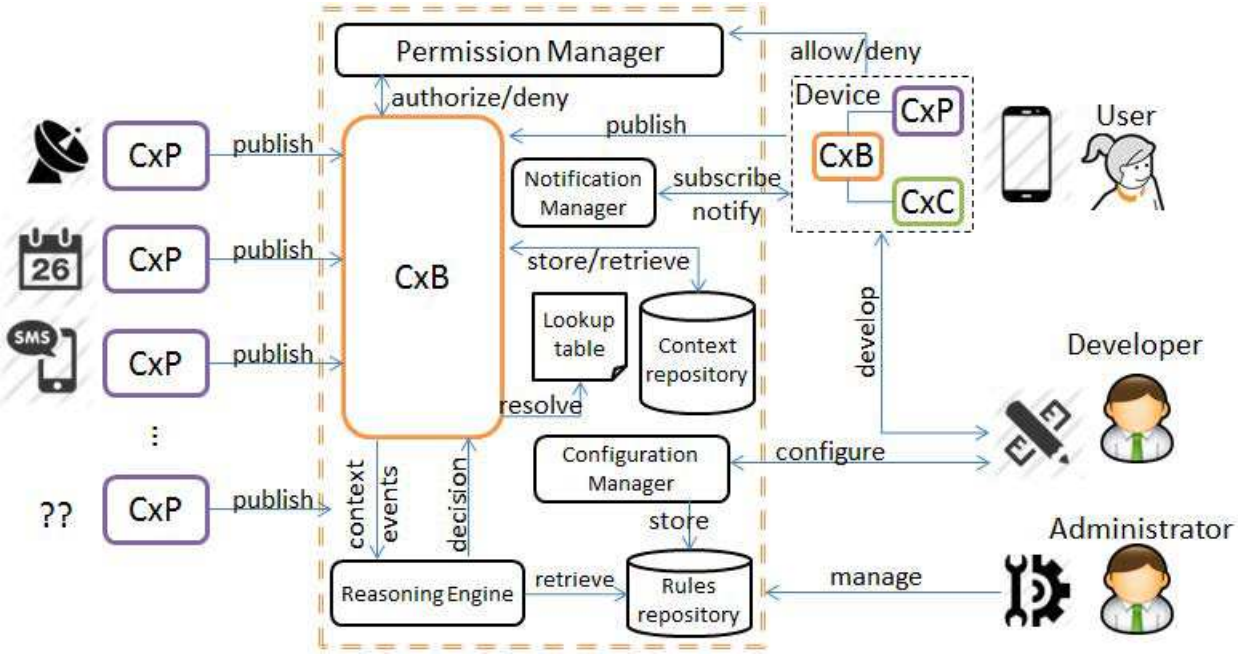

Figure 4.2: Building blocks of the context management system

The user is the owner of the device on which the context-aware application is installed. The developer is in charge of developing the context-aware application in addition to the definition of the application's reasoning configuration. The latter is then uploaded to the CM component that instantiates the corresponding reasoning rules after analyzing and validating the configuration file by checking it against the corresponding definition file(s). After instantiation, the rules become operational and the RE component will apply them every time a new context event is received. The administrator is in charge of managing the infrastructure, for instance migrating the databases or deploying the platform on new servers.

4.1.3 Operational view

From the operational viewpoint, the different tasks and operations performed by the context management system are depicted in figure 4.3. These operations may be performed at different moments of the life cycle of the CMS. The design time concerns the definition of the configuration files that describes the context management functionalities (i.e. abstraction, aggregation, etc.). At the runtime, the third-party components connected to the context management platform can play two roles: the role of Context producer or the role of the CxP. In some cases, a given component may play both roles if needed. CxPs are the source of context information, they provide context in two modes:

- Push mode: the CxP push context updates to the platform in a regular way for basic providers; or under a specific situation for more intelligent providers.
- Pull mode: the CxP exports a specific interface that the platform can invoke in a synchronous way to request an update.

Context consumers represent the sink point of context; an example of such components may be an application that consumes context data to display an aggregation of them to the end-user in a convenient way on a dashboard.

In addition, the runtime moment involves the execution of the operations already defined at design time of the context-aware application which are triggered for execution upon receiving context updates from the providers. The result of these operations may be forwarded out to consumers that may have shown interest in this result through a prior subscription.

Another operation concerns the administration of the CMS that can be performed at any time of the CMS lifecycle to configure and manage the other operations. For instance, it provides an interface for context-aware applications developers to upload the configuration files defining how context should be processed for a given application.

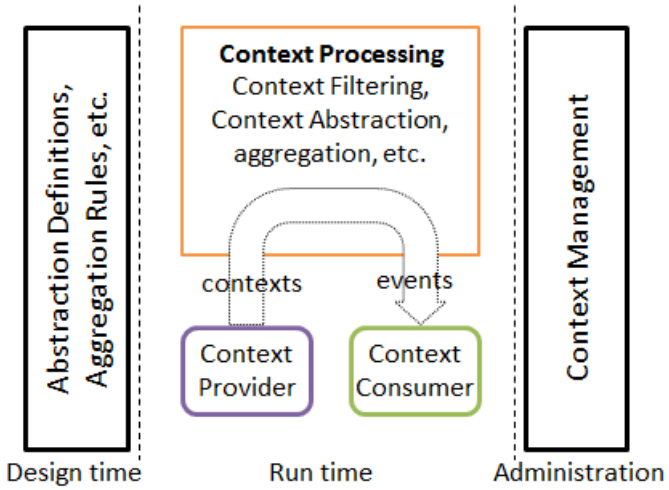

Figure 4.3: Operational view of the context management

4.1.4 Scaling for the Cloud

The component responsible of the context processing in the platform represents the bottleneck of the whole architecture as it receives huge amount of information from different sources, process them and send out notifications. Thus, a scalable architecture should be designed in an efficient way to face peaks in information amount that came for instance from Machine to Machine (M2M) applications.

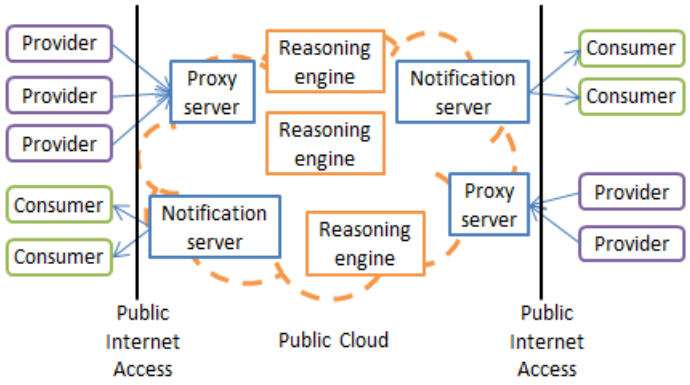

Figure 4.4: Architecture of the cloud-based context management platform

Figure 4.4 presents a cloud-based framework for managing contextual information at a large scale. The architecture expands the main components of context

management framework into smaller components implementing basic functionalities (e.g. sending notifications) and elastic enough to scale on demand. Providers (CxP) are the components responsible of wrapping source of context and publishing it to the ecosystem. Consumers (CxC) are implemented by context-aware applications that request context to adapt its behavior to changes in the user situation (set of contextual information). The Proxy Servers (PS) are the cloud gateway; they receive context updates and publications from CxP and perform load-balancing by routing these updates to the available reasoning engines. Also, they receive from context-aware applications a description file of their adaptation rule that will be hosted by the platform. Reasoning Engines (RE) are responsible of instantiating these description files in order to implement corresponding context-reasoning process. This reasoning results in a set of events to be sent to the corresponding application in order to adapt its behavior in response. Notification Servers (NS) support asynchronous protocols (e.g. Comet [1], WebSocket [2]) to callback and notify CxC about new published context or with events generated by a reasoning engine after processing context. The callback happens on the reference defined by the application in its description file. All communication messages are handled with standardized protocols: HTTP for synchronous communication; Comet for asynchronous communication and event notifications. The cloud components (PS, RE and NS) are implemented as separate RESTful web services that may scale efficiently.

The sequence diagram of the exchanged information between the different components of this cloud-based architecture is depicted in the following figure 4.5. Reasoning engines have to register with a proxy server in order to be solicited later for implementing reasoning processes. Application developers upload to the proxy the description file of their application context reasoning process. The proxy server then chooses a reasoning engine to host and instantiate the reasoning process. A simple Load balancing mechanism based on Round-Robin is used by gateways to choose a registered reasoning engine that will process the new published context. Context-aware applications subscribe with the notification server. When a context is published by the context provider to the proxy server, the later forward it to the corresponding reasoning engine to process it and may trigger an event to consumers through the notification server. At any time, developers can upload a new version of the description file of their application reasoning process and thus adapting its context-awareness to meet new requirements.

1. http://cometd.org/
2. http://tools.ietf.org/html/rfc6455

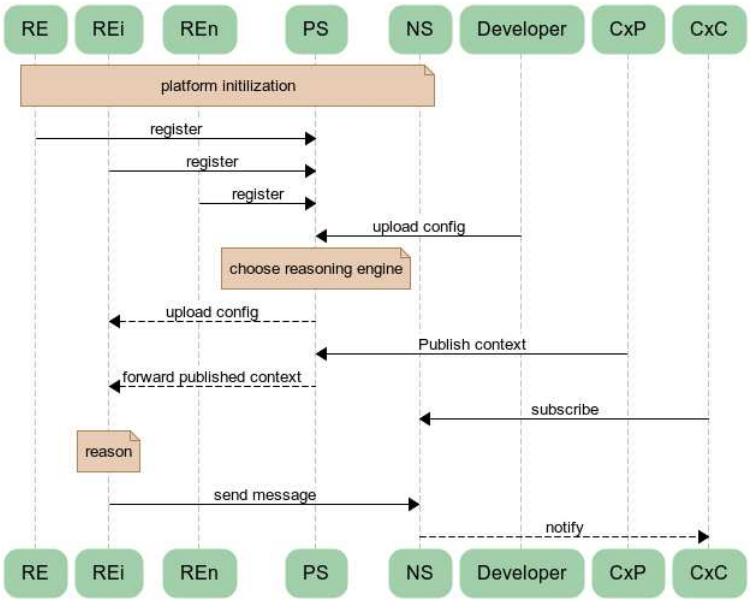

Figure 4.5: Exchanged information between different components of the platform

4.2 Context representation

The different definitions of context which are well established in the research community of context-aware computing are broad and generic as they intend to cover the general ideas of the community on what is context. In addition, they share a common property which consists of having an entity-centric view of context. In fact, context was always seen as an element or an attribute associated with a single entity. For example, the GPS location is a variable that the values are attributed to the entity located at this position. The entity-centric view is illustrated in Figure 4.6.b that depicts each entity as a star surrounded by contextual information that describes this entity situation. Contextual information are not interrelated, for example the user activity depends highly on time and location information. Furthermore, context usually describes a connection between two entities, for example in case of interpersonal communications. Consequently, and relaying on Dey et al. [7] definition, it is more natural to consider context as any information that can be used to describe a relation between two different entities or the same entity with a reflexive relation. Thus, by considering the entities as nodes, a graph can be constructed given a more global view on the environment of the context-aware application than it will be if each entity was considered separately. The context graph represents a context-centric view of the application knowledge; it is illustrated in figure 4.6.a.

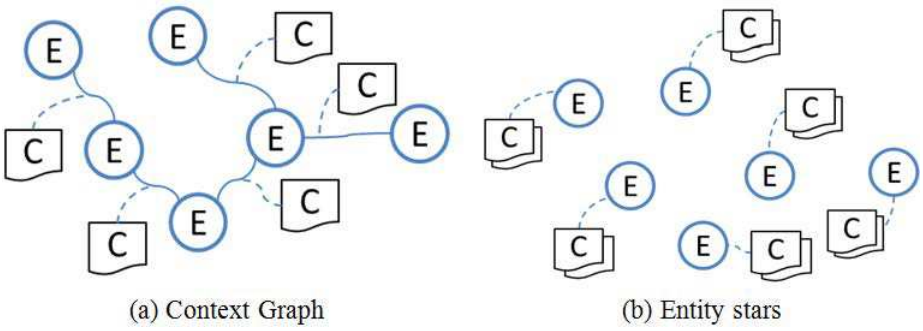

Figure 4.6: Context-centric vs. Entity-centric view

For the realization of the context graph, RDF tuples can be used to represent the connection between two nodes where the relation is represented by a predicate (e.g. isLocatedIn) and the entities referenced in the subject or object of this tuple. An example of a context relation represented in RDF would be (Alice, isLocatedIn, office); here the predicate is used to represent a relation between two entities. In most of background works [1] RDF predicates were used not to represent relations between entities (i.e. the context-centric view). But instead the predicates were used to represent the type of relation (which means the semantic behind the relation) between the source and target nodes of an RDF tuple like (ContextData, isContextOf, Entity) as depicted in the following figure 4.7. This choice has strong implications on what context reasoning technique can be integrated in the system.

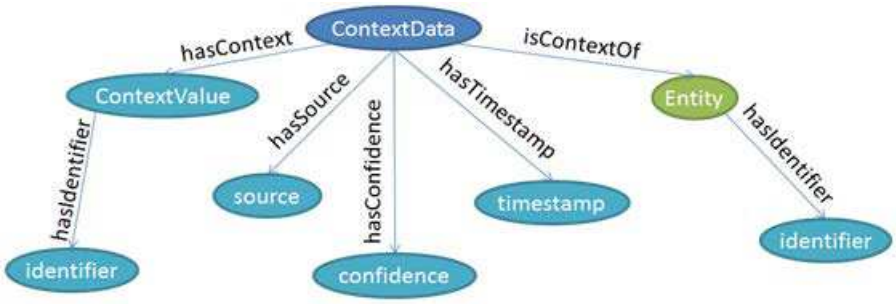

Figure 4.7: Example of an entity-centric representation [1]

An example of a context graph with a context-centric view is depicted in the following figure 4.8 where the connections between two users, Alice and Bob, with each other and with their physical environment are represented. These users are in a phone call and thus can be related to each other with a 'communication'

relationship. In this communication, Alice is using her softphone installed on her computer; she has a 'presence' relation with her softphone. Alice's softphone is connected to Bob's phone throw a SIP session. Both users are also connected to their environment; Alice has a 'location' relationship with her office, and Bob with a meeting room. In addition, Bob has an attribute that designates his workload which is not necessarily represented as a relation between two nodes as it concerns only one user. Workload information can be used by a Call management system to decide how to handle incoming calls (e.g. reject incoming calls when workload is high).

Figure 4.8: Example of a context-centric representation

The context-centric view enables the consideration of the lifecycle of a contextual relationship and the support to the relation evolvement through different states. For instance, the 'Session Initiation Protocol (SIP) call' relation (the following figure 4.9 illustrates the corresponding state transition) is first initiated, then established, and finally terminated. For the 'presence' relation, possible states are online, offline, busy. We call context event the transition between two context states that may trigger a certain action. Handling context events is very important for a context-aware application not only to keep a consistent representation of the world as proposed by earlier works, but also to provide real time adaptation and context-awareness. For instance, a context-aware communication assistant needs to intercept the initiation of a communication in order to provide a context-aware management of this communication.

In addition, the context events can also be used as an efficient support for information dissemination as they can be spread asynchronously to third party applications to trigger an action.

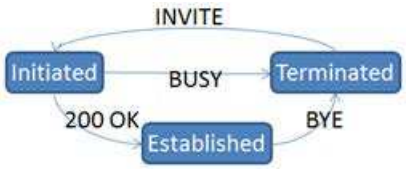

Figure 4.9: Call state transition in a SIP environment

4.3 Context reasoning

The context reasoning aims to generate new information of a high relevance to the application and/or user starting from a variety of context data. It represents the intelligence part of the context management platform that process context in the aim of feeding the adaptation layer with derivative knowledge that helps for accurate decision making. The following subsections describe the concept behind the context reasoning procedure.

4.3.1 Abstract-Aggregate

The Abstract-Aggregate framework aims to represent the inner intelligence of the platform as a network formed by a set of basic functions that can be programmed by external components or a developer to deliver a custom context reasoning functionalities. The network is activated upon some specific context data are published to the context management platform. This activation lead to process this new update as well as other persisted context data previously published in some cases. The update is propagated from a node (representing a function) of the network to another forming a path. Each time a data is passed between nodes an event may be triggered to activate another path in the network and thus enabling a complex information processing. Eventually, this complex behavior may result on a notification to be sent to a specific application that had already subscribed for it.

Figure 4.10 illustrates the network of context management with sample connections between the different functions of the context management system. The description of these functions is further elaborated below: The “produce” function consists of producing raw contextual information. It is implemented by context providers that wrap sensors to comply with the framework API for publishing context.

The “filter” function provides signal processing functionalities that aim to eliminate or at least reduce noise in contextual information and that are collected as physical measurements. It is implemented via the Filter element that defines

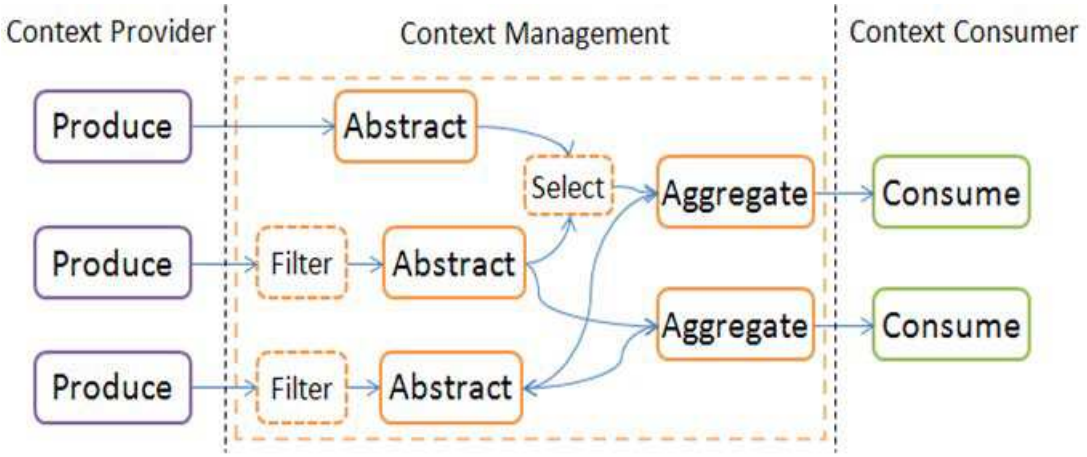

Figure 4.10: Illustration of the context processing network

which signal processing algorithm to use for any given contextual information available from the corresponding source. Some examples of signal processing algorithms that can be selected are the Kalman Filter and the Simple Moving Average (SMA), which is a type of Low-pass filter. For example, a moving average filter can be applied to the returned values of a temperature sensor to smooth the returned values and eliminate possible noise.

The "abstract" function transforms raw contextual information into a higher level of abstraction. It is implemented via the "Automata" element that defines a finite state machine composed of the different states that context data may have. As a trivial example: the context 'availability' may have two states, 'available' and 'not available'. The context 'availability' transits between these states following changes occurring at a raw context level, such as 'presence', sensed from an instant messaging system. A finite state may transit from one state to another only if some preconditions are satisfied. These preconditions are related to context publication from outside (i.e. from CxPs), to internal events triggered from another finite state machine, or are the result of timeout expiration when the condition is implemented as a timer. A transition may lead to events, which can be internal (to trigger another finite state machine or an aggregation rule), or external, such as to notify a context consumer of its callback address.

The "select" function enables the selection of the 'best' available context (from among those provided by multiple sources) based on programmable criteria. It is implemented with the "Select" element that defines the parameter to compare in the selection process, and that describes the operation to apply for this selection process (i.e. choosing the maximum or the minimum value). The compared parameter might characterize the quality of a context, such as accuracy or precision.

For example, the abstract location (e.g. home, work) of a person can be obtained by abstracting GPS information and also from location information acquired from an instant messaging system. The difference between the two location sources is in the precision, which is high for the first source and low for the second. A tracking application may choose to select a source that provides the maximum available precision.

The "aggregate" function performs an aggregation on a set of contextual information in order to generate a composite context to be exposed to context-aware applications. The "aggregate" function is implemented thanks to the 'Rule' element that defines a set of conditions to be met by different contextual data, and a set of actions, in the form of events and notifications to a context-aware application. The aggregate function represents an IF-THEN rule that can be triggered by one or more abstract functions under special states, i.e., when some pre-conditions have been verified. The condition part of an aggregation rule is composed of a set of key-value pairs where keys represent context data (e.g. availability) and values represent references or thresholds related to this context data (e.g. available). The verification of the condition part is performed after that the rule has been triggered.

Finally, the "consume" function is performed by context consumers (i.e., the context-aware applications) as they consume the context of any level (raw, abstract or composite) and adapt their behavior accordingly.

4.3.2 Configuration document

The functions (i.e. abstract, aggregate) defined in the previous section when combined create a complex network of context processing where the connections between processing nodes establish the flow of information inside the context management platform. The way how developers can define the basic functions as well as the connections between them is through a well-structured specification document.

Figure 4.11 depicts the sequence diagram related to the management of reasoning configuration document. It illustrates the communication between the components from the subset of the context management system which is involved in the context processing. After designing the application, identifying the reusable components including the CxP and consumers and implementing the missing components, the developer specifies the configuration file that will describe the interactions between the developed application and the context management system. The CM (Configuration Manager) is the entry point for the developer, it receives from him the configuration file to check its validation, initialize the corresponding reasoning that will be used to process context updates and make decisions on the generate knowledge. The reasoning rules are stored

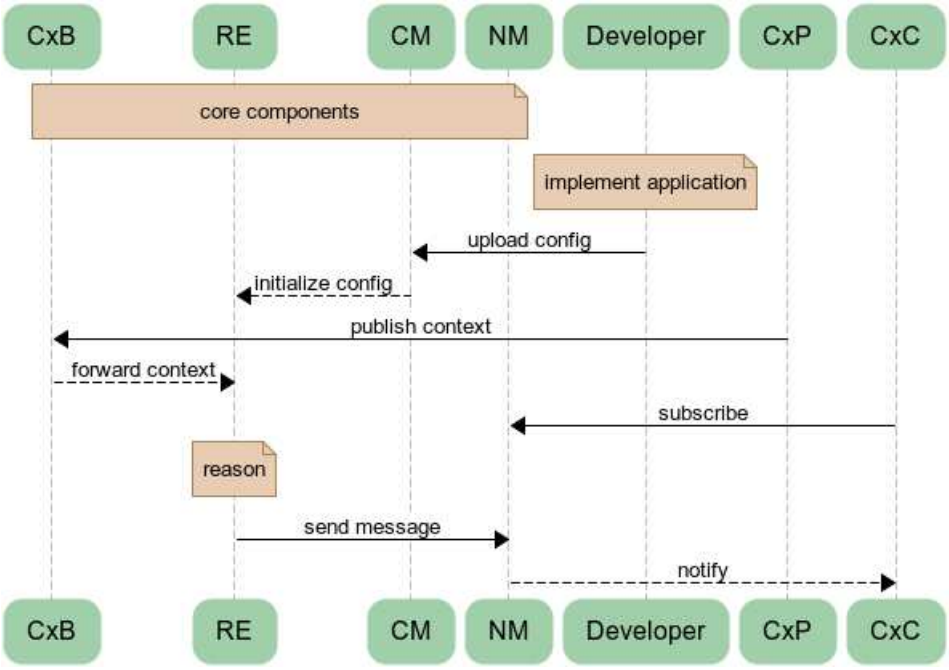

Figure 4.11: The communication between the different architecture components

until the RE (Reasoning Engine) loads them on a specific context event to initiate the context processing. The CxB is the endpoint exposed to CxPs: it receives context updates from the latter, generates a context event and forwards it to the RE. The latter proceeds the processing of the context event in combination with other contextual information. The result of the operation may potentially be sent to a concerned application via a notification through the NM (Notification Manger).

The specification document, upon received by the management platform, continuously evolves over time as a result of actions being performed on it (e.g. creation). The different states in the lifecycle of the configuration document are as follow:

- Undefined This is a baseline state. The context management platform does not know anything about configuration files in this state as they are not yet defined or created.
- Defined defining a configuration file aims to make it manageable by the platform, for this it should conform to the Document Type Definition (DTD) file defined earlier. The developer can modify easily any file in this state as they are not yet instantiated or running on the context management platform.

– Saved - The configuration file has been uploaded to the context management platform and instantiated so that it can be used for context processing. In addition, already running documents that have been suspended can be in this state.
– Running As a result of a new context update, the corresponding configuration file has been created and started either as transient or persistent domain. Either domain in this state is being actively executed on the node's hypervisor.

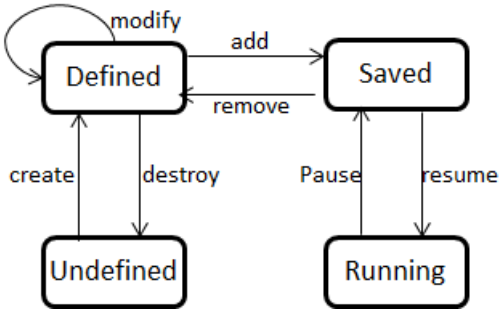

Figure 4.12: life cycle of the configuration file

The diagram above 4.12 shows how markup document states flow into one another. The rectangles represent the different states of the document, and the arrows show the actions that can be performed on a document to move it from one state to another.

4.4 Privacy Management

Controlling which element is accessing what information is a key to protecting user privacy since there is automatic provisioning of user contextual information. Figure 4.13 represents a snapshot of the holistic architecture (depicted in figure 4.2) that concerns only the different components involved in the management of the user privacy.

The role of the permission manager (PM) is to allow the users of context-aware applications to define permissions on a per-element basis, i.e. for each consumer and provider involved in the context-awareness process. It consists of a repository for storing users' privacy policies, and a privacy Policy Decision Point (PDP) for evaluating and issuing permission decisions for CxCs requesting access to a given context. In addition to a brokering role, the CxB also acts as a privacy Policy Enforcement Point (PEP) and is responsible for ensuring the enforcement of decisions made by the PM based on each user's policies regarding how his/her

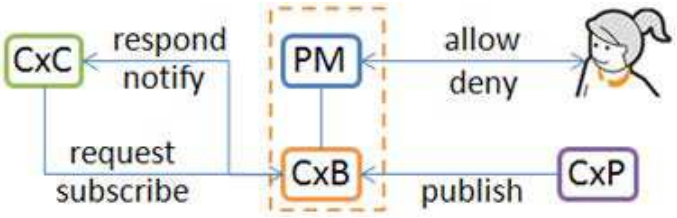

Figure 4.13: Context-aware privacy management

context data are to be used. Users are asked, through a web portal, to give their permission (allow) or not (deny) for the execution of certain operations related to the management (production, storage and consumption) of their context data. These permissions are stored as privacy policies in the PM repository. At any time, users can view their current privacy policies and modify them. Users can also discover which component is accessing their context data, as well as the details of the information provided. For example, users can see the context consumers that are using their location data and the latest updates sent, or the last-requested information. In a future version, we plan to implement an email notification to users whenever a new context consumer appears. The infrastructure is able to provide this information thanks to the use of channels (i.e. context URI) for addressing context, and lookup tables for associating components to a given context as producers or consumers. If there is a policy forbidding a CxC from requesting a specific context, then no new publications corresponding to this particular context will be forwarded to that CxC.

Context	Allowed		Denied	
	Read	Write	Read	Write
Presence	CxP1, CxC3	CxP1	-	CxC3
Location	CxC1, CxC3	CxP2	CxP2	-
Activity	CxC3	CxP1	-	-

The availability of information about how contextual information is used by the different components helps to best select and enforce privacy-related decisions. In fact, users can easily decide how to set their privacy policies and modifications at runtime, e.g. after observing some not-so-suitable behavior from a certain component. This privacy manager can be used as a technical foundation for simple user interfaces.

4.5 Types of reasoning

The proposed framework can be used in many different ways, which can be classified into one of the following categories based on how the reasoning is performed: at the application or the broker side, or distributed across both sides. These different categories are presented in the following sub-sections with representative applications.

4.5.1 Hosted reasoning

In the hosted reasoning approach, the context-aware application hosts the reasoning on its side and the framework is only used for connecting and facilitating the communication between the different parts (i.e. CxP, CxC) of this application. An example of such an application is HEP: enhanced un-interruPtability (Chihani et al., 2011b; Chihani et al., 2011c). This sample application generates workload information related to the use of communication tools (SMS, phone) for users by collecting and processing communication logs. Then, instead of sharing presence information, HEP shares workload information with the user's contacts. Such information helps a user to assess whether his/her contacts can receive additional communication requests or not. HEP consumes the logs histories and processes them locally on the user device, and then relies on the framework to dispatch the generated information to all the other subscribed instances. When a new HEP instance is installed on the user device, this instance registers itself to the broker (CxB) to declare the channel where the workload of the corresponding user will be published. Each time this user adds a new contact to his/her address book, HEP will send a request to the broker to subscribe to the workload information of this contact. As a result, the broker context look-up table is updated and each publication of the newly-added contact's workload information is forwarded to the subscribed user. In this scenario, the benefit brought by the use of the framework consists of reducing the effort needed for connecting several instances of HEP to circulate the workload information of the corresponding user in real time.

4.5.2 Hybrid reasoning

In the hybrid approach, part of the reasoning is performed on the broker side and another part is performed on the application side. For the part made at the broker side, the application has to upload its reasoning configuration file that will be instantiated by the broker. As an example of such application, we present Context-Aware Address Book (CAAB), an android application designed to enrich users' address books with contextual information representing their contacts'

availability. Availability statuses (green for "available", red for do not disturb, orange for "busy", and yellow for "unknown") are generated by aggregating a set of raw contexts, namely:

- The presence information provided by a communication suite (Microsoft Lync);
- The indoor location, which is information provided by a Bluetooth proxy; and
- The workload information related to communication tools (SMS, phone), that are collected from the smartphone communication logs (as described in the previous section).

CAAB was developed for business users to help them contact their colleagues without disturbing them. This application also provides a means for users to rapidly send notifications to their peers in the form of predefined messages augmented with contextual information (e.g. a sender's location). For example, in figure 4.14, a notification message augmented with the indoor location of the sender is displayed to the receiving user. The moment of the notification delivery is conditioned by the receiver's actual workload information, i.e. the notification will be delivered only if the receiver's workload is not equal to "do not disturb", which refers to highest degree of 'busyness'.

Figure 4.14: Screenshot from CAAB

The part of the reasoning performed by the application concerns the generation of the workload information. However, it is the broker that is used to combine this information with presence and location information. Figure 4.15 depicts the deployment of the different parts of the application, as well as the instantiation

of the context reasoning on the broker side. In this particular case, the reasoning aims to generate availability information that corresponds to a single user, Alice. It is composed of two finite state machines that perform the abstraction of a set of context data (location and workload) and a single aggregation rule to describe user availability. Presence information does not need to be abstracted as it already has an abstract nature. Regarding this configuration, a user is considered to be available if he/she is in a free' state (i.e. has a workload rating that is less than 50%), has a presence status 'online' on a given communication channel (e.g. instant messaging), and is located at the 'office'. Otherwise, a user is considered to be unavailable. This is a simple way to derive availability information from other contextual information. We can easily change how availability is derived by modifying the corresponding CPDL structure. Also, reference values used in this Context Processing Definition Language (CPDL) (e.g. threshold 50 for workload level) can easily be personalized for a given user. These customizations can be performed without having to modify an application's behavior.

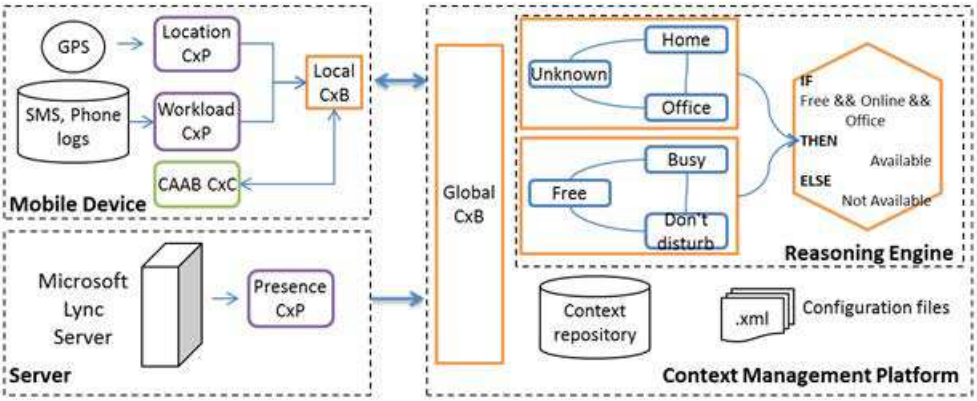

Figure 4.15: CAAB context management

In this scenario, the framework allows the segregation of the application into multiple independent parts and their deployment across different locations (devices or servers) without introducing any complexity regarding the exchange of context data.

4.5.3 Delegated reasoning

With the delegated approach the reasoning is performed on the broker side and the application only receives messages containing the action that should be performed. One example of an application that uses this approach is an IP Multimedia Subsystem (IMS)-based context-aware communication system (Chihani

et al., 2012) that considers user context and preferences in handling incoming calls (e.g. accepting or rejecting a call).

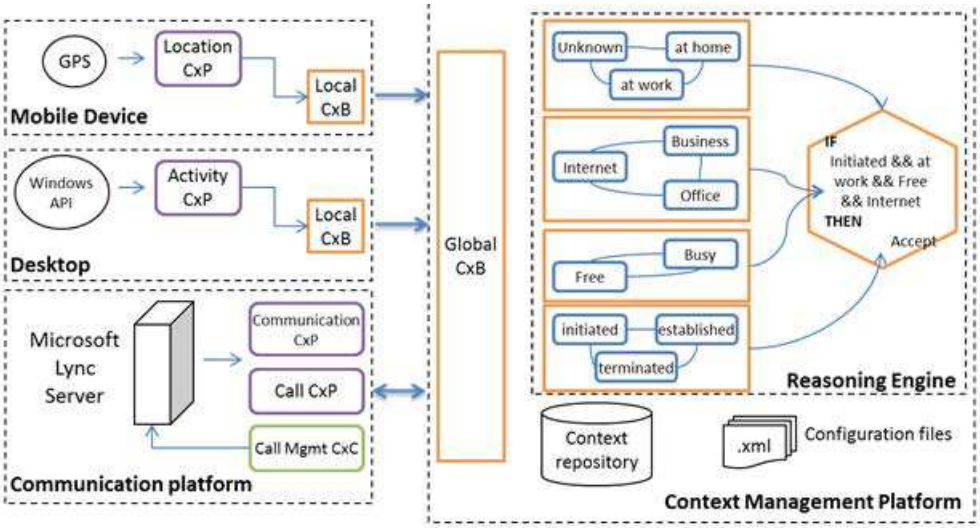

Figure 4.16: Context-aware call management

Figure 4.16 depicts the deployment of the different parts of the call management application. It also shows an example of a call management rule where user context is modeled as abstraction state machines, and its preferences for handling incoming calls are implemented as aggregation rules. Multiple types of contexts are used in the management rule. User location is obtained from the location provider and then abstracted to two main locations (work and home); other locations which are not labeled are abstracted as unknown state. The activity context is related to the active application (e.g. document processing software, client's email application) currently in use. This context is abstracted to the type of application: office gathers applications like MS Word or Outlook, business gathers specific applications such as Customer Relationship Management (CRM), internet is related to internet browsing. The communication context is related to communication activities, whether the user is in communication or not. Information from the call provider corresponds to the state evolution of an incoming call; when a call is initiated it triggers the aggregation rule that will check the current status of the different state machines to decide how to handle that call. The action resulting from the aggregation rule will be sent to the call management component for execution. In this scenario, the framework allows a significant simplification in the development and maintenance of applications as a result of separating the context management (collecting it from sources and reasoning on it) from the core functionalities of the application, here the control

of communications.

4.6 Conclusion

4.6.1 Discussion

This section discusses how the proposed context management system address the design considerations previously introduced in chapter 2. The proposed context management system provides mechanisms to support the development of both kinds of applications. The publish/subscribe communication paradigm is implemented to allow applications to subscribe to a stream of events and be notified upon new information are published. The first class of applications with a simple subscription can be notified by all updates, however the second class will need to provide selection criteria when it subscribes.

The proposed context management system support code reusability by providing developers with an SDK that hides complex and common tasks related to context management. It also provides a basic kind of scaffolding by allowing developers to describe some operations related to context processing in an XML-based language.

The architecture of the proposed context management system is a hybrid architecture that takes benefit of both kinds of architecture. In fact, a global central component is used to provide high level management operations, for instance connecting the different parts of a same application as well as parts of different applications. A local manager is used to add another level of centralization to the architecture in order to avoid unnecessary direct interactions with the global manager. This component is in charge of caching local information to make it available to local components and thus it limits the direct requests to the manager of the higher level. In addition the global component can be deployed in a cloud platform so that it scales on demand to avoid having a bottleneck.

The proposed context management system relies on XML as the markup language used for describing applications at design time. A dedicated component of this context management system is provided to manage the mapping of markup elements into operations and/or method calls. The latter may concern components from the context management system as well as other software components deployed as part of the same application or other third-party components. The attributes of the markup element should indicate the necessary information needed by the manager to correctly instantiate this element and establish the needed connections with external components. The framework hides the implementation details concerning how the communication between the coupled components is really performed.

4.6.2 Summary

The core contribution of this chapter lies in the introduction of a software architecture that can be applied to the creation of applications from different domains. This will be also illustrated later in the case study chapter. The proposed software architecture relies on the analysis of different class of context-aware applications taken from several domains (e.g. communication, assistant applications) as well as on their functional decomposition which were both elaborated earlier in chapter 2. The functional and layered decomposition of the proposed software architecture facilitates the integration of existing applications to enhance them with context-awareness features with the least possible effort. In addition, it facilitates the communication of a unified understanding of the common components and operation of context-aware applications between the different actors involved in the creation of such applications.

Another contribution of this chapter lies on the reconsideration of the established definition of context and proposes a new representation for this concept that will be of important benefit to the design of context-aware applications and the modeling of context, especially for social networking applications.

The subsequent chapter will present the realization of the different concepts introduced in this chapter into a development kit, and also illustrates different implementations of using this support tool for the creation of context-aware applications.

Chapter 5

Implementation

The conceptual framework for context management presented in the precedent chapter provides a considerable support for the creation of context-aware applications by hiding and abstracting common and complex context-related operations. This chapter describes the implementation of these concepts, it goes further in detailing the internal parts of the implementation and how the development support is exposed to developers. The organization of this chapter is as follow: in section 5.2 we described the core components of the context management system which is the Context Software Development Kit (SDK). This SDK is based on the well-established layered architecture presented earlier in the previous chapter. It offers a set of ready to use components and guidelines for assisting developers and enabling a rapid development of context-aware applications. In fact, this SDK offers for each layer a separate package that holds specialized components for handling complex operations related to this layer. The section 5.2.1 illustrates main classes used in the implementation of the providers' layer which are responsible of accessing the sources of context information. The context representation is introduced in section 5.2.2 through the different classes of the presentation package that provides flexible and powerful tools for dealing with context information. The adaptation package described in section 5.2.3 where is gathered the representation of the different components responsible for performing complex processing operations on context data. Last section (i.e. section 5.2.4) talks about the package corresponding to the consumers' layer which includes components responsible for performing the adaptive behavior.

5.1 Context SDK

This section introduces the proposed context management solution and describes in details the realization of this solution through an SDK with the corre-

sponding different libraries.

5.1.1 Overview

The conceptual Context Management System presented in the previous section aims to facilitate the development and maintainability of context-aware applications by hiding complex technical details (e.g. adaptation support) from developers and providing ready to use software components that abstract common functionalities (e.g. sensors provisioning). Furthermore, this system can be integrated to legacy systems like Information Technology (IT) systems in order to extend them with context-awareness capabilities. The following figure 5.1 depicts the different parts of the implementation of this conceptual Context Management system in a form that follows the layered architecture presented in chapter 2. The main parts are the Context SDK that offers a ready to use software for building context-aware applications, the main systems of an Enterprise IT as well as systems of the communication platform.

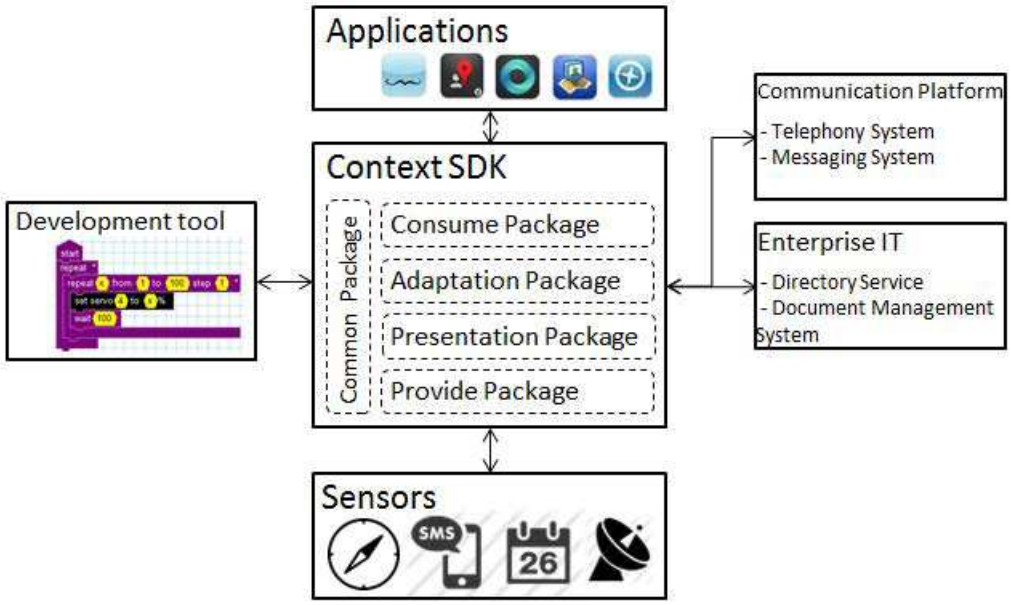

Figure 5.1: Components of the Context Management System

The services of the Enterprise IT represent the source of information that the context management system relies on. Example of information gathered includes profile information about the employee enterprise (e.g., full name, manager, identifier), meta-information about the managed documents (e.g., authors, contributors, key words). While the underlying services of the communication

platform represents the targeted services that the context-aware application under development aims to enhance them by making them more aware of the user situation. For example, an adaptive behavior from the target context-aware application may cause a communication service to pop-up a notification to user for an incoming call saying that it is an urgent one, or simply reject the call without even disturbing the user. The context management system provides means for controlling the execution of these services by acting on their configuration or the set of parameters provided on their invocation. Example of the services includes VoIP communication system like Microsoft Lync or messaging systems like Outlook. The core part of the context management system is the Context SDK that provides developers with a set of library that they can use while developing their context-aware applications.

5.1.2 Development Support

A SDK is provided to ease the developer task; it represents the core component of the context management system. This SDK extends the Java Development Kit (JDK) with additional libraries that implement main functionalities of the Abstract-Aggregate framework (figure 4.10). It follows a component-based software engineering approach to provide standardized software components that encapsulate common context-aware applications features (e.g., context acquisition). Furthermore, this SDK provides an additional support for context-aware applications development and maintainability by providing XML binding to enable the instantiation of components from an XML configuration document (and vice versa). To implement context-aware applications, third party developers need add this SDK to their development environment and extend its cores classes. The remainder of this section details the UML diagrams describing the implementation and the usage of this SDK.

Provider package

Providers represent the source of context data that will feed the whole context management chain to end-up performing adaptation. In the SDK, they are represented with software components that wrap specialized sensors like hardware sensors or web services. These components are part of the context.sdk.provide package (depicted in figure 5.2). Developers have to extend the BasicProvider class to implement a specific provider able to manage an underlying sensor (i.e. receive data from it, perform a cleaning, etc.). A provider can supply context data to other components synchronously in a request-response fashion through the provide() method. It can also provide it asynchronously in a publish/subscribe way by providing the addListener() public method for subscriptions and notifying

them later when data is available by calling the notify()method that will dispatch an event to be captured by any classes implementing the IProvideEventListener interface. The provider must also implement the getQuality() method to provide information regarding the quality (e.g. accuracy, error rate) of the underlying sensor; these information will be needed later by the adaptation layer, for example to choose the best sensor that fit a certain quality requirements. The SDK provides some ready to use providers to access common sensors available in Android-based devices (e.g. GPS, WiFi, battery, communication log), as well as web service (e.g. RESTful service).

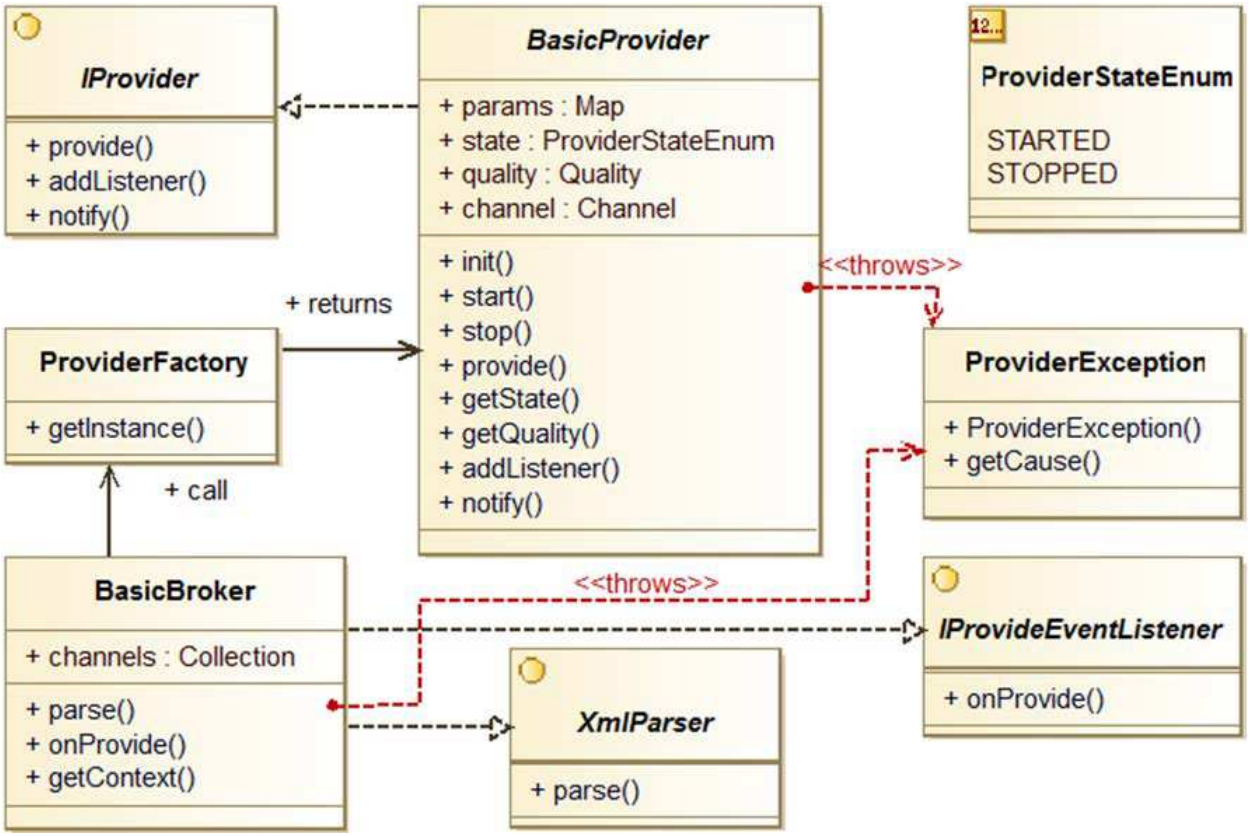

Figure 5.2: UML diagram of the provider package

The SDK provides XML biding for automatically instantiating providers based on a given XML specification through the ProviderFactory class. A ProviderException is triggered in case the system encounters an error while parsing the XML (e.g. due to mal formatted XML) or the provider failed to start using the provided configuration parameters. The listing 1 depicts an example of an XML document that can be used to describe the initialization parameters of a provider instance.

The identifier and class attributes are mandatory, for the address attribute, in case it is not provided in the XML document, the default value is localhost. The broker and entity information are also mandatory, they enable to associate the provider with the context management infrastructure. The association of the instantiated provider with an entity enables to match the published context data with its owner entity. The association of the provider with a broker enables the provider to report gathered context information to the right components. The XML specification may contain additional information for example for custom providers not provided by the SDK but implemented by third party developers. Examples of implemented context providers for two the Android platform include a GPS location provider, and a communication (SMS and phone calls) provider. The SDK do not provide means for auto-detecting or registration of already deployed providers.

```
Listing 1: Example of XML specification for instantiating a location provider.
<providers>
   <provider identifier='gps_p39'
      class='context.sdk.provide.GPSLocationProvider'
      address.ipv4='192.168.56.1'>
         <broker type='local' address.ipv4='192.168.56.1' />
         <broker type='global' address.ipv4='192.168.56.1'
          address.port='9090'/>
         <entity identifier='abcd1234'/>
   </provider>
</providers >
```

Presentation package

The context model defines the way how the context data should be represented in a way to reduce the complexity of sharing context among the context-aware application components. The context.sdk.presentation package provides core classes for representing context data and other objects that may be relevant to context management. Figure 5.3 illustrates the main classes of the presentation package. The model is flexible in a sense that it provides developers with the ability to extend the Context class to represent domain-specific context information (e.g. motion information). But developers should keep the information hold by the subclass consistent and do not overload it for example by including preference information with location information in the same class. A context instance is multi-dimensional and it is represented with a set of typed attributes that associate a key to its value. As the SDK is based on Java, the attribute types can be primitive types like:

- Integer/double to represent physical measurements or to hold a counter value,

– Boolean to represent binary data (e.g. absence of a feature),
– String to hold a sequence of characters (e.g. Unicode characters),
– Date to hold time information.

The attribute can be also of a complex type that the developer had already defined.

The Context class implements the IEvaluable in order to be able to evaluate the context information hold by an instance against another one which is important to evaluate context-based conditions.

The distribution of context data is based on the use of channels which are instances of the Channel class. A channel allows to uniquely identifying a context data instance of a certain entity. As the full architecture is based on REST principles [77], context information are considered as resources that can be referenced by the framework components (i.e. providers, brokers and consumers) via a URI (Unified Resource Identifier). The CxP chooses a channel (e.g. /abcd1234/presence) on which it will publish the produced context (e.g. presence information). Providers of the same contextual information (e.g. location) can publish content on the same channel. The CxB contains a mapping table that matches context data and corresponding URIs for requests and subscriptions to context, and another lookup table (table 5.1) is used to track who is publishing to and who is consuming from a given context channel. A consumer (CxC) can request or subscribe to specific contextual information by contacting the CxB to start listening on the corresponding channel and be notified when new information is published.

Table 5.1: An example of a CxB context lookup table

Channel	Context Providers	Context Consumers
A= /{user_id}/presence	CxP1	CxC3
B= /{device_id}/settings	CxP2	CxC1, CxC3
C= /{location_name}/location	CxP1	CxC2

Inspired from the blackboard communication model [78], the channels are used as a communication way for an efficient sharing and reduction of dependency among the different components involved in the management of an instance of context data (i.e. providers, brokers, consumers).

In the same way, the interface between the provider and presentation packages is guaranteed thanks to the ContextFactory class. This class is called by a provider to create a new empty instance, and then thanks to the populated methods of the Context class, the provider can manipulate the returned context object. The factory class can also be used to create a context object from a markup document like the one in listing 2.

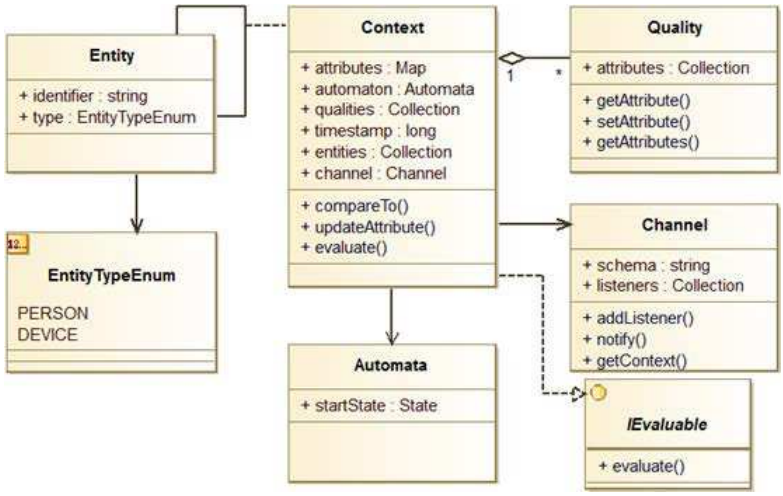

Figure 5.3: Context meta model in UML

The Entity class represents any supervised entity (e.g. user, device) for which there is a set of providers that publish its related context data. The context class represents an association class [79] that describes and maintains information about the relationship between two entity instances. This modeling approach allows the representation of the context graph which relies on considering entities as nodes and on transforming context information to links that join two entities.

The SDK also supports, through the Quality class, the representation of meta-data that may be used to provide additional information about context data. Example of meta-data that may be used to represent the quality of context information includes: frequency of context updates, precision or value range, etc.

```
Listing 2: Example of XML specification representing location information
<contexts>
   <context type= 'location' subtype='gps'>
      <meta timestamp='12345' expiration='12347'/>
      <latitude>33.376439</latitude>
      <longitude>6.862087</longitude>
   </context>
   <context type= 'location' subtype='address'>
      <meta timestamp='23561' expiration='23680'/>
      <street>27 rue aouis</street>
      <city>el-oued</city>
      <postal>39000</postal>
      <country>algeria</country>
   </context>
</contexts>
```

The use of channels in conjunction with an XML-based context representation provides a hierarchical addressing schema allowing the composition of context data. For instance, if Alice's presence information is published on channel '/alice/presence' and her location information is published on '/alice/location', then Alice's composite context information (presence and location) can be requested from channel '/alice'. In the same way, if location information is represented in XML and contains an element for GPS coordinates (longitude and latitude) and another element for civil address, then channel '/alice/location/gps' should return the GPS coordinates while the channel '/alice/location/address' should return the civil or postal address.

In order to provide an interface between the adaptation and the presentation packages to support interactions between software components of the two packages, each instance of the Context class is connected to an Automata instance. The later represent a finite state machine that tracks any changes of context and transform it to state transitions. These transitions may be may lead to triggering and adaptation process. The subsequent section will describe in details the adaptation process which is based on finite state machines.

Adaptation package

Context data are collected by applications to maintain an accurate vision about the user situation in order to respond with an adaptive behavior to any context changes. Here an adaptive behavior means not only taking into account the user situation to execute an action but also means that the action performed by the application correspond to a need of the user to the result of the action. The SDK provides support for defining situation of interests and also describe the corresponding adaptive behavior through the different classes (depicted in figure 5.4) of the context.sdk.adaptation package.

The aim of the adaptation layer consists of a first consumption of context updates propagated by the presentation layer, reason about these changes to filter any situation of interests. Then, it may forward the context update to any consumer who may had declared its interest to this particular situation in order to handle performing adaptive behavior. Figure 5.4 illustrates the core classes of the adaptation package as well the bridging components that relate this layer with the previous and subsequent layers. The root class of this package is the Control class which holds an aggregation of the different classes responsible for performing the adaption process as proposed by the Abstract-Aggregate paradigm. As described earlier in section 4.3.1, this paradigm relies on four different functions: filter (to eliminate noise on context data), abstract (to generate a higher knowledge from the raw context), select (to choose the best source of information) and aggregate (to compose context into a situation). In the SDK, the filtering function is repre-

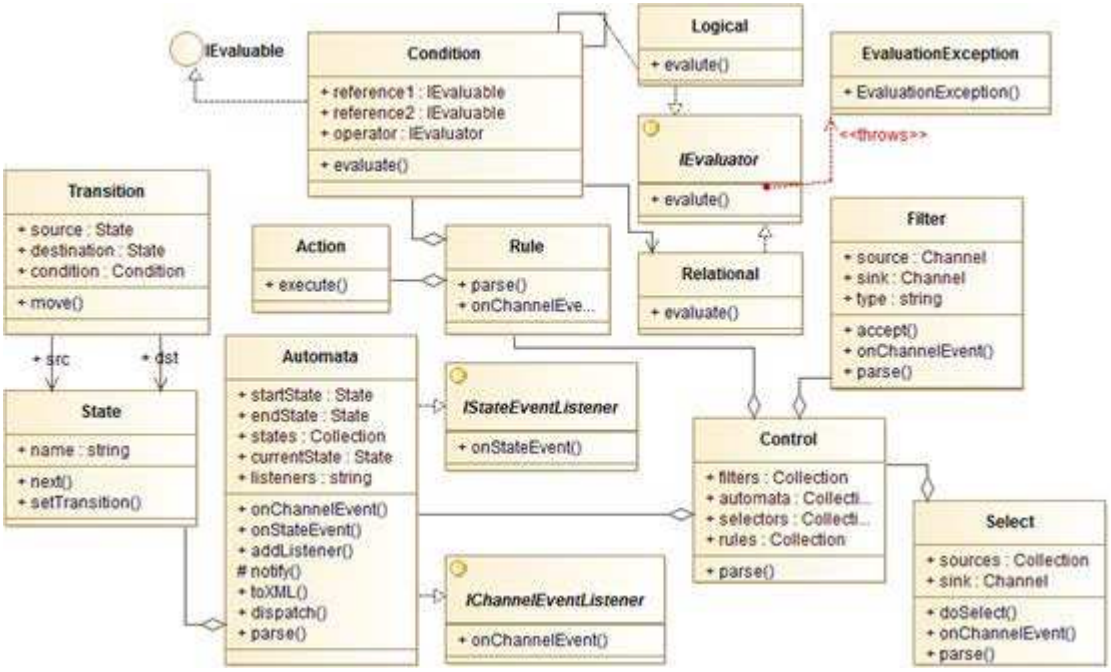

Figure 5.4: UML diagrams of the adaptation layer

sented by the Filter class which basically provides an accept()method to consume context updates submitted on a source channel (i.e. context address) and send the filtering result to a destination channel which may trigger the invocation of another function. The SDK provides some basic filtering types like Simple Moving Average (SMA) [80]. This class can be extended by developers to create custom filtering functionalities on custom context data. The abstraction function is represented by a finite state machine with an UML meta-model inspired from [81]. The main classes of this finite state machine are the Automata, State and Transition classes. The finite state machine is used to perform abstraction by mapping raw context information, represented by the Context class, to a higher knowledge, represented by the State class. This process is triggered by a context update and captured by the Automata class in its onChannelEvent() method. The different states are linked through instances of the Transition class to organize how the high level information (modeled by states) evolves in response to context updates. The selection function is represented by the Select class provides capabilities for selecting context data from multiple available sources to a unique destination channel based on context quality criteria. Finally, the aggregation function is represented by an IF-THEN rule modeled in the adaptation package with a Rule class. The latter holds a combination of Condition instances

to represent a complex Boolean expression that should be satisfied and a set of Action instances which will be invoked when the rule is satisfied. Two Condition instances can be combined with a Logical operator (e.g. and, or) which will regulate how the combination is evaluated. Also a Condition instance may compare a Context instance to a State instance with a Relational operator (e.g. equal, not equal) to check whether.

Consumer package

Consumers are important components of the context management chain located in the consume layer; they are responsible for realizing changes to the environment by mapping the decisions taken by the control layer to real-world actions. They can be tiny programs deployed on end-user devices responsible for performing a unique task and having a limited lifecycle or complex web services with constraining availability requirements able to run long operations. The Context SDK offers, through the context.sdk.consume package, some specialized software components that can be used by developers to build custom consumers able to perform complex functionalities. Furthermore, the SDK provides mechanisms for implement consumers with active (i.e. automated actions on behalf of user based on context updates) or passive (i.e. displaying contextual information to end-user) context-aware behavior. Some build-in consumers include functionalities for the management of incoming calls (e.g. rejecting) on mobile devices as well on an IMS [82] (IP Multimedia Subsystem) environment, video streaming on mobile devices, as well as displaying webpages.

Figure 5.5 depicts the main classes of the consume package, BasicConsumer class is the root class of all consumers implemented by the SDK or the ones that will be implemented by third-party developers. ConsumerFactory is a Singleton class [55] used to instantiate a consumer and initialize its basic parameters by parsing an XML configuration file. The main parameters of a consumer include the address (represented by a Channel instance) of the context information from which the consumer should request or receive context updates, and the context broker to which is associated this consumer.

```
Listing 3 : Example of XML specification for instantiating a consumer
<consumers>
   <consumer identifier='droid_call_c51'
      class='context.sdk.consume.android.CallMgmtConsumer'
      address.ipv4='192.168.56.1'>
         <broker type='local' address.ipv4='192.168.56.1' />
         <broker type='global' address.ipv4='192.168.56.1' address.port='9090'/>
         <entity identifier='abcd1234'/>
   </consumer>
</consumers >
```

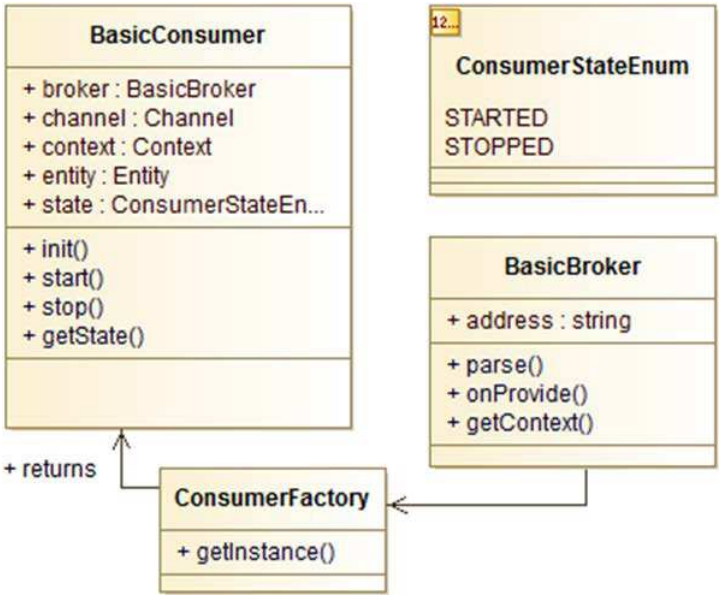

Figure 5.5: UML diagrams of the consume package

The listing 3 depicts an example of a markup document that can be used for the initialization of a consumer. In this case, it is a consumer responsible for managing incoming calls (i.e. able to perform actions on calls like rejecting it) represented by the CallMgmtConsumer class. Similarly to providers, the SDK do not provide means for auto-detecting consumers. As a consequence, it is in charge of the already deployed consumers to register to the platform in case they are custom made or they have to be deployed after their instantiation from an XML specification.

Common package

The common package gathers a set of transversal operations and common utilities that can be helpful in many cases. Some of them are private only accessable by the rest of the other packages, others are publicly exposed and can be invoked directly by the developer. For instance the context.sdk.common.security package offers a set of classes for developers to use in order to control the manipulation of their application data. Since the third-party components are allowed to interact with the platform, it's possible that an untrustworthy component could access to sensitive context information. Choosing the right way to secure the contextual information is important; there may be several ways to do it dependently on the sensitivity of the stored data and the application needs. For the current

implement, the platform offers an unique security enforcement through the use of permissions for accessing and updating context data. The most flexible way to secure context data is through the use of Access Control Lists (ACL) where each context data has a list of components along with their access permission they have on that particular data. A component needs read permissions in order to retrieve a context data, and a write permission to update or delete that data. Developers can also set a default ACL for all of context data created by their application.

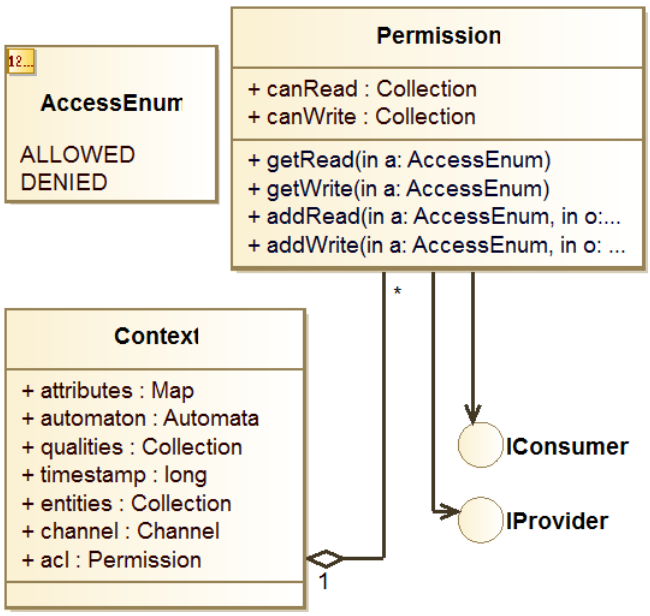

Figure 5.6: UML diagrams of the security package

Figure 5.6 depicts the main classes of the security package; Permission class is the main class for enforcing access control permissions. It stores information about the rights for manipulated data given to a component (e.g. a provider or consumer) that can be ALLOWED or DENIED as specified in the access type enumeration. Theses access rights are then used by the Context class to control the access to each of its instances. This access control paradigm is implemented at the object-level which means the permissions once given are enforced on all type of data stored in an object without distinction. A more fine-grained control can be implemented to enforce access to each attribute separately, however this will considerable complicate the data manipulation with a direct implication on the design of the Context SDK.

5.2 Using and Configuring the Context SDK

The proposed Context SDK provides programming models and abstraction mechanisms to support the development of application adaptation processes. The SDK supports also the maintenance (i.e. control and reconfiguration) of an already defined adaptation process with a minimum effort (i.e. without having to re-implement it). Most of these programming mechanisms are easy to use as they can be described with markup documents which when parsed allow for a domain-specific instantiation and initialization of software components.

This section introduces some general programming mechanisms which are very common in most context-aware applications and can facilitate their development. The first set of mechanisms concern the support for comparing between different context data as well as some cases where it can be used. The second concerns the abstract-aggregate framework that enables the control of the event flow generated from context updates, and the management of context processing operations on behalf of applications. The final mechanism concerns the use of context data to manage graphs composed of entities. These different mechanisms are proposed by the SDK through a dedicated set of classes.

5.2.1 General programming mechanisms

As most context-aware applications relies on rule-based adaptation methods, it is important for them to be able to compare different context information to find out the set of satisfied rules. Context comparing can be useful in many case, for example to find entities showing similarities on one of their context dimensions (e.g. check whether two users are in the same meeting). Also, context comparing can be used to check if one entity's context satisfies a set of criteria (e.g. check if user is in a certain situation like available). Another example of using comparison procedure is for a filtering purpose in order to find entities from an initial set based on certain similar characteristics shared by their contexts (e.g. find a device that best fits the user current situation to forward an incoming call to it).

The Context SDK supports these types of matching procedures through the context.sdk.presentation.comparing package. This library facilitates the composition of Boolean evaluators (e.g. equal) that make the comparison of two context instances possible. Furthermore, these evaluators represent the basic construct for filtering context from an initial set or for triggering appropriate actions based on context information. The following subsections describe with more details the Boolean evaluators; also introduce the configurable filtering functionalities.

Context-based evaluation

This kind of evaluators extend the IEvaluator interface, they are used in Boolean expressions to compare context information of multiple entities. The SDK represents the context evaluators in a tree with context information as leafs or this tree, and nodes in the intermediate levels, as well as the root of the tree, are binary or unary evaluators that concatenate underlying nodes. There are two types of evaluator provided by the SDK: Logical (e.g. and, or) used to combine conditions and Relational used to compare similarities (e.g. equal, less, greater) between two context information. Both have an evaluate() method which returns a Boolean when called to run this evaluator on its leafs.

Listing 4: Example of XML specification for instantiating a consumer

```
<condition type='logical' class='context.sdk.presentation.comparing.AndEvaluator'>
   <condition type='relational' class='context.sdk.presentation.comparing.EqualEvaluator'>
         <operand type='channel' value='/device_1/msg_123/status' />
         <operand type='reference' value='unread' />
   </condition>
   <condition type='relational' class='context.sdk.presentation.comparing.EqualEvaluator'>
         <operand type='channel' value='/device_1/mode' />
         <operand type='reference' value='handsfree' />
   </condition>
   <condition type='relational' class='context.sdk.presentation.comparing.EqualEvaluator'>
         <operand type='channel' value='/device_1/applications/music_player' />
         <operand type='reference' value='active' />
   </condition>
</condition>
```

Listing 4 depicts a sample markup document representing a Boolean evaluator that checks the status of an incoming message, whether the device mode it is hands-free as well as if the music player is the current active application. This composed condition can be used of instance by a rule that aims to decide how to render an incoming message, for example the action part would be reducing the player volume and playing, to the user, the speech that corresponds to the message. The EvaluatorFactory class of the comparison package is used to parse such an XML presentation. This class is also responsible for instantiating the basic evaluators (e.g. and, less) defined by the SDK.

In case the Boolean evaluator referrers in one of its operands to non-standard context information (i.e. implemented by a third party developer in a custom made class) then a custom comparison have to be implemented. The context SDK supports advance comparison procedures by providing the compareTo() method of the Context class. This method takes as parameter another instance of the Context class and has to be over-ridded in the context subclasses.

Context-based filtering

The filtering procedure aims to find, from an original set, a subset of elements that satisfy certain criteria or share similar properties. A typical case where filtering can be used to bring context-awareness concerns an application that aims to recommend new contacts by finding a set of entities that share similar interests to the user. The Context SDK provides support for filtering operations that take as input a collection of entities as well as selection criteria and returns the subset that fulfills the requirement specified by the criteria. The selections criteria are evaluated to true or false and refer to a combination of conditions. The FilteringEngine class of the comparing package is responsible for performing the filtering procedure which iterates over the input set of entities, call the evaluate() method of the current entity with the selection criteria as an input. This method performs the real evaluation and developers must override it to be able to perform custom filtering. In case the evaluation returns true then this entity is added to the filtering result set otherwise it is ignored and the engine jump to the next one until the last entity of the input set.

5.2.2 Context processing management

As the architecture of a context-aware application is most of the time distributed with different components deployed across different locations, it is import to have an underlying communication paradigm which is very flexible to limit the overhead due to exchanged data and their volume. The Context SDK relies on Inter-Process Communication (IPC) [83] techniques to establish communications between lightweight components in the same device. While for heavy components distributed on different machines, the communications establishment and messages distribution rely on protocols and techniques from the Service Oriented Architectures [84] ecosystem. While relying on these complex techniques, the SDK provides a simplified interface to developers so that they focus most of their effort on developing their application and ignore the complexity of context management operations. In addition, with such highly distributed architectures the volume of exchanged messages may increase tremendously even exponentially if the same message should be sent to multiple receivers. The SDK address this issue by providing developers with mechanisms for controlling the rate of messages that should be forwarded to their applications thanks to the Abstract-Aggregate framework. The following subsections describe with more details the usage of the Abstract-Aggregate framework to reduce overhead; as well as how the implementation of the flexible communication paradigm used in the implementation hides events distribution complexity.

The Abstract-Aggregate framework

The Abstract-Aggregate framework provides developers with mechanisms to control the granularity and volume of data from the value stream that should be forwarded to the consumer applications. This approach aims to optimize the event flow generated by continuous context updates sent from sensors. The discretization of a stream of raw context updates through the Abstract-Aggregate framework happens as follow: when a sensor value changes significantly, the corresponding provider fires an event notifying all components that have registered as listeners to this event (i.e. who had implemented the IProvideEventListener interface). After receiving a new context value through a notification, the listener can perform a transformation of this value according to the Abstract-Aggregate mechanisms (e.g. filtering) implemented in the Context SDK or by custom made ones. These components can in return propagate the context update and send a new event with higher knowledge obtained by transforming the initial event, and thus, further reduces the sampling rate of the original context updates stream. When a component responsible for performing adaptive behavior receives these context change events, it triggers the evaluation of its internal adaptation process which is implemented with the Context SDK as a rule system. In case a particular context update leads to the fulfillment of the precondition of a rule, the associated actions are then executed and the context-aware application behavior is adapted. This complex process reduces considerably the overall computation complexity of the context-aware application.

Context processing specification

The SDK provides support for configuring seamlessly the software components of the adaptation package that perform functions of the Abstract-Aggregate framework through an XML-based specification language . The configurability allows developers to modify the behavior of the components of this package with a minimum programming effort not only at design time but also at run time. Listing 5 illustrates the DTD of the language used to initialize and configure the components of the adaptation layer responsible for performing abstraction and aggregation. This DTD is used to validate the syntax configuration document before instantiating any components.

```
Listing 5 : Document Type Definition of the Abstract-Aggregate of XML specification
<!DOCTYPE Definition [
<!ELEMENT Definition (Filter | Automata | Select | Rule)*>
<!ELEMENT Filter (Input, Output)>
<!ATTLIST Filter Type CDATA #REQUIRED >
<!ELEMENT Automata (State*, Start-State, End-State)>
<!ATTLIST Automata Id CDATA #REQUIRED Name CDATA #REQUIRED>
```

```
<!ELEMENT State (Transition)*>
<!ATTLIST State Name CDATA #REQUIRED >
<!ELEMENT Start-State (Transition)*>
<!ATTLIST State-State Name CDATA #REQUIRED >
<!ELEMENT End-State EMPTY>
<!ATTLIST End-State Name CDATA #REQUIRED >
<!ELEMENT Transition (Condition, Event*) >
<!ATTLIST Transition Dest CDATA #REQUIRED >
<!ELEMENT Select (Input, Output) >
<!ATTLIST Select Param CDATA #REQUIRED Opt CDATA #REQUIRED >
<!ELEMENT Input (Channel+) >
<!ELEMENT Output (Channel) >
<!ELEMENT Rule (Condition+, Event)>
<!ATTLIST Rule Id CDATA #REQUIRED Name CDATA #REQUIRED >
<!ELEMENT Condition ((UOP?, SimplCond),(BOP, UOP?, SimplCond)*) >
<!ELEMENT UOP EMPTY (NOT) "NOT">
<!ATTLIST UOP Type CDATA #REQUIRED >
<!ELEMENT BOP EMPTY >
<!ATTLIST BOP Type (AND | OR) "AND">
<!ELEMENT SimplCond (Channel) >
<!ATTLIST SimplCond Operator (EQ|NEQ|GE|LES) "EQ" Value CDATA #REQUIRED >
<!ELEMENT Event (Channel) >
<!ATTLIST Event Type(internal|external)"internal" Message CDATA #IMPLIED >
<!ELEMENT Channel EMPTY >
<!ATTLIST Channel prefix CDATA #REQUIRED suffix CDATA #REQUIRED >
]>
```

Each markup element of this DTD corresponds to a class in the adaptation package and it defines the initialization parameters of the corresponding class. The 'Definition' element is the root element; it gathers the pointers to the underlying elements. The 'automata' element represents a finite state machine (abstract function) that listens for raw contextual information to know when to transit between from one of it states to another one. Certain transitions may cause an internal event that will trigger a local adaptation rule (aggregate function) by publishing information related to this event on a specific channel. This aggregation rule, represented by the 'rule' element, will compare the current context values to a given situation which is defined in the condition part of the rule, and then notify context-aware applications when a match occurs. As these functions are completely programmable, context-aware application developers are able to define and implement a part of their own reasoning logic into a remote and central component (i.e. the context broker). This provides a considerable benefit to applications, since context updates are filtered and part of their intelligence is implemented on the context management framework, enabling the development of lightweight context-aware applications.

Querying context

The SDK provides support for two different models for querying context data which are synchronous and asynchronous. The synchronous model enables to use the interface provided by a software component to send it a query, with potential parameters that may be used to refine the query result, to receive the corresponding response. While, the asynchronous model relies on the publish/subscribe communication paradigm to enable software components to subscribe to a stream of events generated by other components and to be updated with notifications as new events are generated. Also, for one subscription to a stream the subscribed component may receive many event notifications. The Context SDK hides the underlying complexity of these two communication models through the getContext() and the addListener() methods of the Channel class. The first method provides synchronous communication to request context data, while the second method providers support for the asynchronous paradigm. For example, the implementation of these communication models on an Android device relies heavily on Intent [85] messages which are the unique available mode for Inter-process communication on Android devices. Upon making a call to one of the Channel communication methods, an Intent message is forwarded to the local context broker who will register the subscribed for notifications or will reply to the Intent by sending back an appropriate response. The local broker may be asked to forward the request to the global context provide if the operation corresponding to the request cannot be fulfilled locally. In this case, an HTTP communication [86] is established between both brokers to handle the messages exchange.

5.2.3 Context graph management

The context graph is maintained by processing published context to create/delete nodes (i.e. entities) or edges (i.e. relations) between nodes. First, developers upload an XML description file that contains the graph specification and that will be used to create the graph from subsequent published context (e.g. specification of how links between entities should be created from sensed data). Second, to be able to construct the graph (i.e. create links) we need first to create the nodes. As a consequence, applications (represented by CxC) have to declare the nodes (i.e. supervised entities: user, device) so that the engine will create the corresponding node as described in the XML description file. Third, edges creation is performed by the engine that will use published context by sensors (via CxP) to create links between nodes as described in the XML description file. The XML description file can contain specifications for push notifications in the form of IF-THEN rules, where the condition part is an XQuery string that will be executed each time the corresponding graph is updated (e.g. after edge creation,

node deletion), and the action part is a notification message where the destination is a common callback channel on which any CxC can listen, the message body is constructed from the query result thanks to XPath expressions. In case, the XQuery execution returned nothing then no notification is sent out. The representation of the graph is important to be able to provide such functionality. For this we use Sedna, an XML database that provides an XQuery engine, to store the managed context graph.

Graph Creation

Listing 6 presents the DTD of the XML language used to describe how a graph of a given application should be constructed. The root element of this XML file is the Graph element, it has four sub-elements: Registration, Node, one or more Connectors, one or more Notifications.

Listing 6: Document Type Definition of the Context Graph XML specification

```
<!DOCTYPE Graph [
<!ELEMENT Graph (Registration| Node | Connector*| Notification*)>
<!ELEMENT Registration >
<!ATTLIST Registration channel CDATA #REQUIRED >
<!ELEMENT Node (Attribute*)>
<!ELEMENT Connector (Channel | Edge) >
<!ATTLIST Connector id CDATA #REQUIRED type CDATA #IMPLIED >
<!ELEMENT Channel >
<!ATTLIST Channel prefix CDATA #REQUIRED suffix CDATA #REQUIRED>
<!ELEMENT Edge (Attribute*)>
<!ELEMENT Notification (To| XQuery)>
<!ELEMENT To >
<!ATTLIST To startwith CDATA #REQUIRED endwith CDATA #REQUIRED callback CDATA #REQUIRED >
<!ELEMENT XQuery >
<!ELEMENT Attribute>
<!ATTLIST Attribute name CDATA #REQUIRED value CDATA #REQUIRED>
]>
```

The 'registration' element describes the channel that should be used by CxC instance willing to declare the graph nodes. The 'node' element describes how a graph node object should be created after receiving information published on the registration channel. This element has a list of attributes where a name correspond the attribute name (e.g. type, identifier) and the value correspond to the XPath string used to retrieve the real value of this attribute from the published information on the registration channel. The 'connector' element has two sub-elements: Channel, Edge. The 'channel' is used to build the channel on which the engine should listen to receive context publication that will be used to connect between nodes via this Connector. To create the channel, we need a common prefix and suffix, the cardinal is the identifier of nodes. The

'edge' element is used to connect between nodes, it has a list of attributes that correspond to the tag of an edge and the corresponding XPath string used to retrieve the value of the tag from published context. The 'notification' element has two sub-elements: To (describe the callback on which the notification message should be dispatched), and an XQuery string to be executed on specific events (e.g. publication on a channel that have similar prefix as the one declared in the 'To' element) that leaded to updating the graph.

Graph Representation

A graph (or network) is a data structure composed of vertices (nodes) and edges (relations). The following examples demonstrate different graph modeling (that may serve different purposes) for the same scenario where a group of people are interconnected with each other through social connections or interconnected to objects from their environments. In this scenario, the graph is constructed from collecting and abstracting contextual information. Figure 5.7 depicts a context graph where a person can be connected to another person through social relations, for example the 'has contact' relation means that the original person has declared the second one in his address book. In addition, a person can be connected to an object through a physical relation, for example a person can be connected with a room through the 'is located' relation that means that this person's physical location corresponds to the room identified by this object. Also, two objects can be interconnected, for example the non-direct 'proximity' relation links two rooms with each other and means that the physical locations are approximately not far. The semantic behind the relations depends on the application interpretation, for example the 'proximity' relation is an abstraction of the physical distance and as a result may be interpreted differently by two different applications.

For the same scenario, the listing 7 illustrates an example of how the context graph can be encoded in order to easily store it and process it. The graph is represented in a GraphML-like language where node and edges can have any number of attributes depending on the need for the given application.

```
Listing 7: An example of an XML representation
<graph id='contacts_graph'>
   <node id='1234' label='Alice' type='Person' />
   <node id='5678' label='Bob' type='Person' />
   <node id='9012' label='Chuck' type='Person' />
   <node id='3456' label='office_128b' type='Room' />
   <node id='7890' label='room_129' type='Room'/>
   <edge tag='has contact' from='1234' to='5678' isDirected='true'/>
   <edge tag='follow' from='1234' to='9012' isDirected='true'/>
   <edge tag='is located' from='1234' to='3456' isDirected='true'/>
   <edge tag='is located' from='5678' to='7890' isDirected='true'/>
```

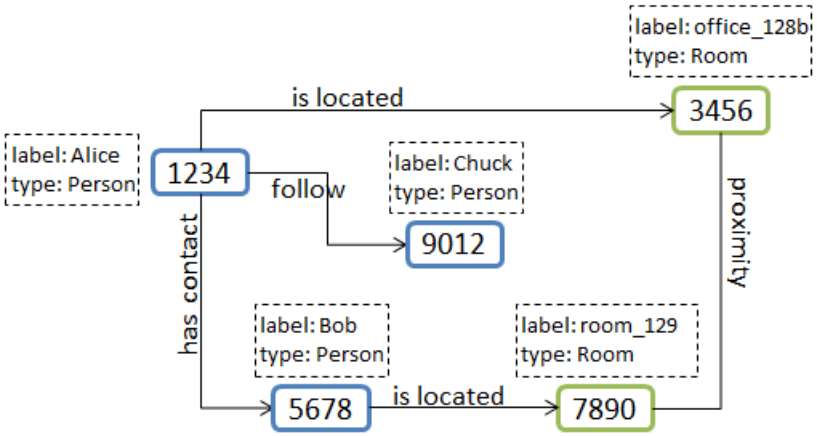

Figure 5.7: An example of a graphical representation

```
   <edge tag='proximity' from='3456' to='7890' isDirected='false'/>
</graph>
```

Graph Traversal

Applications can send polling requests to monitor the state of the graph by performing a graph traversal. This operation consists of a step by step graph exploration based on hops; it may be used for example to get the neighbors of a node, or to find paths between nodes. Figure 5.8 illustrates how a traversal is performed: starting from an initial node (step 1) to a final node (step 4). At each step, the current visited node is depicted in red. A path ending on a node without any outgoing edges that fulfill the traversal criteria are abandoned as depicted in step 3. At the end of the traversal process, the nodes being currently visited constitute the result of the traversal which will be encoded in an XML representation in order to send to the original application that submitted the query.

Listing 8 illustrates an example of an XML graph traversal query that returns all followers of people who follow same people that an initial user is following.

```
Listing 8: An example of a traversal query
<traversal graph='twitter_graph' startfrom='me'>
   <outedge>
      <condition key='tag' operator='equal' value='follows' />
   </outedge>
   <innode assign='followed' />
   <inedge>
      <condition key='tag' operator='equal' value='follows' />
   </inedge>
```

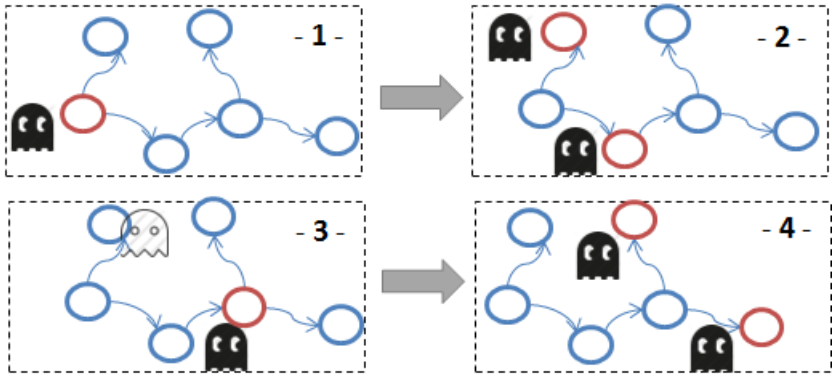

Figure 5.8: Traversal-based graph reasoning

```
   <outnode except='me' />
   <outedge>
      <condition key='tag' operator='=' value='follows' />
   </outedge>
   <innode except='followed' />
</traversal>
```

In this XML, the 'traversal' element has a 'graph' attribute that represent the identifier of the graph on which the query will be executed, the 'startfrom' attribute designates a node of the graph from where the traversal should start. The 'outedge' element allows selecting all outgoing edges from a given node that satisfies underlying condition. A condition consisting of comparing an attribute of the element (here edge) with a reference value using an operator (e.g. equal). The 'innode' element asks for selecting nodes that have an incoming edge from the edge selected in the previous step. For caching temporary a result, this element may have an 'assign' attribute that create a list of nodes from the result returned by 'innode', or the 'except' attribute that remove any node that has been assigned to the named list from the returned result. The 'inedge' element allows selecting incoming edges to a given node that should also satisfy underlying condition. The 'outnode' allows selecting source node of a given edge.

Graph Reasoning

As the graph is stored in an XML representation, the framework natively supports another way for traversing and reasoning on the context graph through the use of XQuery code. The XQuery language as a well-established XML standard provides useful means for the extraction and manipulation of any data represented in an XML document. It also provides a mean for constructing XML document

which is an interesting feature for building an XML representation of the result that will be sent back to the requesting application.

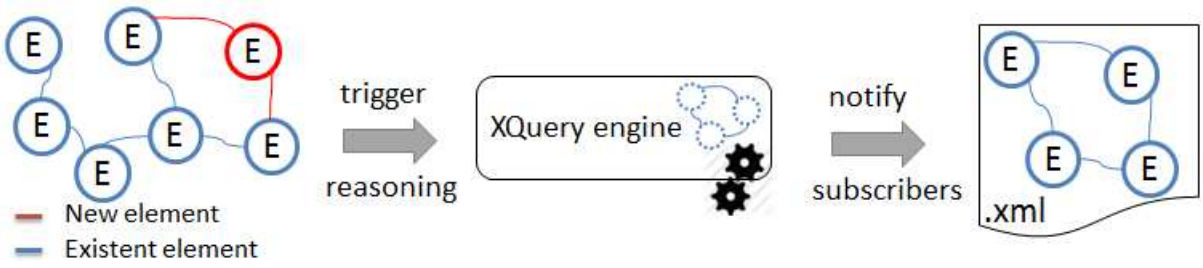

Figure 5.9: XQuery-based graph reasoning

Thanks to these advantages, the use of XQuery provides the ability to perform asynchronous requests. These queries aim to explore the context graph at each update in order to find interesting structures (e.g. a set of nodes linked through a particular relation) in the graph and to send the result to the application that registered this XQuery code. Figure 9 illustrates a typical example of the XQuery execution after an update in the graph structure as well as the generation of a sub-graph encoded in XML and representing the query result. The following listings **??** depicts an example of XQuery code that aims to find the contacts of a user who are located nearby.

```
Listing 9: An example of XQuery code
let $graph := doc("contacts_graph.xml")/graph
for $user in $graph/node/@id
   let $ulocation := $graph/edge[$user=@from and @tag="is located"]/@to
   for $contact in $graph/edge[$user=@from and @tag="has contact"]/@to
      let $clocation := $graph/edge[$contact=@from and @tag="is located"]/@to
      for $proximity in $graph/edge[($clocation=@from and $ulocation=@to)
                                    or($clocation=@to and $ulocation=@from)]
      return {$user} is in proximity of {$contact}
```

The ability to register XQuery code for an execution each time the graph is updated provide the ability to find interesting structures in the context graph. Although XQuery is one of the important standards related to XML technologies, the code writing in this language is barely understandable. Thus an alternative should be proposed to be able to write comprehensive asynchronous queries.

5.3 Summary

In this chapter we provided the description of the proposed context management system that aims to facilitate the main tasks related to developing and

maintaining context-aware applications. In fact, this proposal provides support for construction, integration and administration of the adaptive behavior of a context-aware application.

For easing developer tasks regarding the implementation of the adaptation behavior, the section 5.1.2 presented some appropriate general programming mechanisms that can be used to build context-aware applications. Section 5.2.1 presented some mechanisms based on context comparing that can be used to trigger an appropriate action, also this section discussed how a combination of Boolean operators are used to construct a complex comparison expression. Section 5.2.2 introduced the Abstract-Aggregate framework as a means of custom processing of context through an event-condition-action model, also providing support for the discretization of the stream of events generated by the changing values of context. Furthermore, this section discusses how the Abstract-Aggregate framework can be used to maintain the context-aware application by providing developers mechanisms for reconfiguring the adaptation procedure without having to modify entirely their application. The last section, i.e. section 5.2.3, talks about the context graph and the different procedures related to its management, as well as how it provides support for an explicit representation of relationship between entities by using context information.

To prove the general applicability of the introduced framework for context management, an examination of the concepts detailed in this section as well the introduced context SDK should be considered. The subsequent chapter introduces some case studies where this context management framework was used.

Chapter 6

Case studies

This chapter evaluates the thesis proposals through perfromance evaluation in a simulated environment and also the implementation of different case studies. Section 6.1 performs a set of simulation-based evaluations to examine the benefit of the abstract-aggregate concept which was introduced in section 4.3.1 and implemented in the Context SDK in section 5.2.2. This section attempts also to undertand the scalability characteristics of the architecture and how well the response time of the platform evoluates. Section 6.2 conducts an evalution based on case studies to illustrate how the Context SDK (introduced in section 5.1.2) with its programming abstractions (presented in section 5.2.1) are used to facilitate the construction of adaptive context-aware applications. From these case studies, the derivation of a set of common requirements for an adaptive behavior allowed a practical evaluation of the different adaptation mechanisms provided by the Context Management System. This chapter is concluded with a summary of the lessons learned from the implementation of the different case studies, as well as a discussion of the pros and cons of the approach proposed in this thesis.

6.1 Performance

This section evaluates the concept of abstract-aggregate in a simulated environment where multiple publishers send continuously context updates that will be forwarded to subscribers. The benefit of using abstract-aggregate is to reduce the amount of unnecessary updates that will be forwarded to subscribers through the use of many filtering functions defined by the application developer. The section also evaluates the performance metrics like the response time and delay introduced by the context management platform and its functionalities.

6.1.1 Efficiency

To evaluate the benefit of abstract-aggregate based reasoning compared to a regular case of the publish-subscribe paradigm where no reasoning is performed, we conducted an experimental study based on OMNeT++ [1], a powerful discrete event simulation toolkit. The topology used in the simulation is composed of a context broker to connect to the rest of the components, $n = 12$ CxPs and $m = 8$ CxCs. Context providers continuously publish context messages to the broker in an exponential distribution with a configurable mean (for each provider). Each context consumer subscribes to a random number of providers in a uniform fashion (all providers have a similar chance for a given subscriber $p = 1/n$). The context broker stores all the subscriptions in a routing table; when it receives a context update from a provider it duplicates the message to all the corresponding subscribers. The following table 6.1 illustrates the simulation setup parameters.

Table 6.1: Simulation parameters

Providers	Rate (publication/s)	Consumers
CxP_1	40 (p/s)	CxC_3, CxC_8
CxP_2	20 (p/s)	CxC_1, CxC_3, CxC_7
CxP_3	30 (p/s)	CxC_3, CxC_5, CxC_8
CxP_4	20 (p/s)	CxC_2, CxC_3, CxC_5,
CxP_5	20 (p/s)	CxC_1, CxC_3, CxC_4, CxC_5, CxC_8
CxP_6	10 (p/s)	CxC_1, CxC_3, CxC_5, CxC_7
CxP_7	10 (p/s)	CxC_1, CxC_3, CxC_4, CxC_5
CxP_8	50 (p/s)	CxC_2, CxC_8
CxP_9	20 (p/s)	CxC_1, CxC_2
CxP_{10}	40 (p/s)	CxC_1, CxC_3, CxC_5, CxC_6, CxC_7, CxC_8
CxP_{11}	20 (p/s)	CxC_1, CxC_3, CxC_5
CxP_{12}	20 (p/s)	CxC_1, CxC_3

The graphs in figure 6.1 present the throughput (number of context messages sent or received per second) on the context broker side. The red graph, at the bottom, represents the throughput at reception, i.e., the incoming context updates from the different providers. The graph in green, the upper graph, represents the throughput at emission when the abstract-aggregate reasoning approach is not deployed (i.e., all incoming messages are sent out to all subscribers). The blue graph, in the middle, represents the throughput at emission when the abstract-aggregate method is used to filter incoming updates and only send out relevant events. The throughput at emission is greater than that at reception because

1. http://www.omnetpp.org/

there is more than one subscriber for each context provider, and thus each incoming message has to be duplicated as many times as there are subscribers for the given context information.

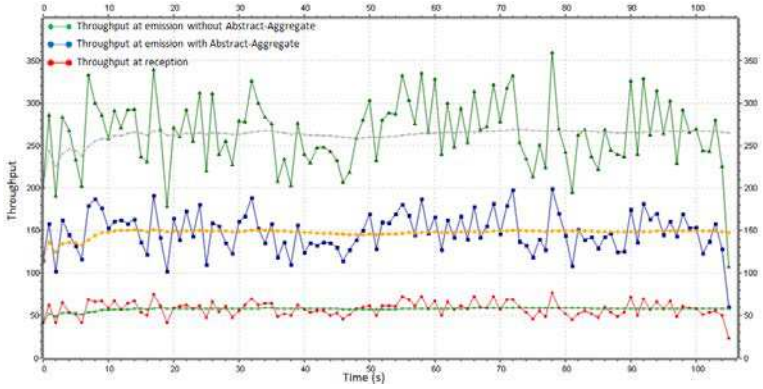

Figure 6.1: Throughput comparison graph

As illustrated by these graphs, the use of the abstract-aggregate reasoning approach allows the reduction of emission throughput, since only relevant events (those relevant to a given application) are sent out. Each context update from a provider is first consumed by the abstraction function that updates the current state of the corresponding finite state machine. The transition between the states may trigger, with a probability p, an event to the aggregation function that will verify the current situation, and as a result may trigger a notification to a corresponding context consumer with a probability q. Thus the probability of an updated context to generate an event out to a consumer depends on both probabilities p and q. The relation between the throughputs when the abstract-aggregate is used and the throughputs performed when the abstract-aggregate is not used is expressed in the following equation:

$$Throughput_{with} = Throughput_{without} * p * q \tag{6.1}$$

The results of the efficiency test suggest that the proposed framework significantly reduces the bandwidth consumption, as less context updates are circulated to consumers. However, these results represent the ideal case assuming an optimized context management configuration file. This may not be the case if the XML documents were not optimized, for instance if any context update would always satisfy the conditions of the aggregation rule so that all context updates will be forwarded to the consuming application.

6.1.2 Scalability

Enterprise scenario

Another concern is to keep the framework response time reasonable to support scalability of the architecture and to support an increasing number of providers and/or consumers. From the context consumer, the response time corresponds to the amount of time needed for the provider to publish context to the broker plus the time needed for the broker to reason on the published context, plus the time needed for the broker to send a notification to the consumer. The transportation time (for publication and notification) depends on the network conditions and cannot be controlled. It is the reasoning time which is more interesting to study as it depends on the quantity of managed context and the number of hosted configuration files. To evaluate the overhead added onto the context broker when using the abstract-aggregate reasoning approach, we used an example of a context-aware system: the CAAB application described previously in section 4.5.2.

Figure 6.2 depicts the deployment architecture for CAAB. The central server runs a Microsoft Windows Server 2003 SP2 (Vendor: Intel, Model: Xeon, CPU: 2.8GHZ, memory: 2 GB). A Jetty web server is installed on the server to run the Context Management Platform (CMP) web applications on top of the RESTlet framework. CometD is used as a notification server to send context updates to subscribed clients. For data persistence, contextual information is stored on an object database, DB4O.

Figure 6.2: CAAB deployment architecture

Microsoft Lync is the unified communication solution. The Presence Provider is a Unified Communications Managed API (UCMA) 3.0 application that monitors the Microsoft Lync users' presence information. Both run on a single 64-bit Microsoft Windows Server 2008 R2 Enterprise SP3 (Vendor: Intel, Model: Xeon, CPU: 2.0GHZ, and 2 GB of memory). The Location Provider is an application

that can detect nearby Bluetooth devices with the help of the Blucove [1] library. The Identity Provider is a web application that provides a directory service: mapping between the physical (Bluetooth) address of a device and the owner identifier, as well as mapping between the SIP Uniform Resource Identifier (URI) used by Lync and the user identifier. Both providers run on Microsoft Windows XP SP3 installed on a Dell D630 laptop (Vendor: Intel, Model: Core 2 Duo, CPU: 2.2GHZ, memory: 2 GB). Figure 6.3 depicts a comparative summary of reasoning time distributions as a ratio of the number of CPDL configuration files uploaded to the context management framework. This figure utilizes box plots, a convenient means of data visualization that is especially useful for detecting the presence of extreme values in a distribution.

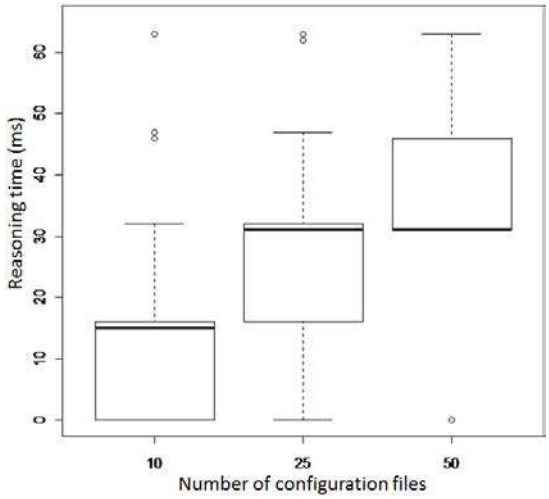

Figure 6.3: Distribution of Abstract-Aggregate reasoning time

The different distributions are generated by sending approximately one hundred sample context publications to the framework and measuring how much time the framework spent processing each publication through the functions' Abstract-Aggregate. In a box plot, a darkened line is used to represent the median (50th percentile) of a distribution, the 25th percentile corresponds to the bottom side of the corresponding box, the 75th percentile is indicated by the box's top side, the distribution's 10th percentile corresponds to the box's bottom line, and the 90th percentile corresponds to the box's top line. Additional points or lines on the top or bottom of a box correspond to the distribution outliers, i.e. points showing the extreme values for a given distribution.

These box plots show that the median for 10 configuration files is about 15

1. http://bluecove.org/

ms, while it is about 30ms for 25 and for 50 configurations, indicating a possible convergence in reasoning time that should be confirmed by further studies. The spread (the box height or the difference between the 75th and 25th percentiles) of the different distributions is very similar for each configuration grouping. Also, except for a few outliers, the reasoning time increases as the quantity of configuration files being managed increases.

The graph in figure 6.4 illustrates the variation in response times, in milliseconds, of the framework for multiple context publications. Data used to generate this graph are collected by sending multiple context publication requests to the framework and measuring the time to receive a response for each request. This response time includes the time required to establish the HTTP connection as well as the time needed for the framework to deal with this published information.

From the figure, we can easily see that the response time oscillates around 150ms; reaches a maximum of 400ms in extreme cases and a minimum of 50ms for the best cases.

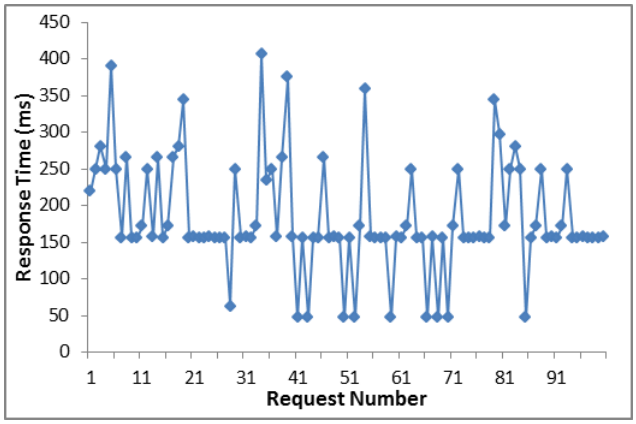

Figure 6.4: Variation of response time

We can conclude from figures 6.3 and 6.4 that the response time of the proposed framework depends mostly on the time required for exchanging HTTP messages among the different elements of the framework. The reasoning time is minor compared to the connection time. An important cause of this latency in HTTP connections is due to the repetitive phases of TCP handshakes. The framework performance, as far as context transportation, can be improved by the use of HTTP-persistent connections. The use of permanent connections will improve memory and CPU utilization related to the establishment of HTTP connections, as fewer connections will need to be opened and maintained. Permanent connections will also enable the pipelining of multiple-context exchanges, and especially of context publications, since the frequency of context collection at the

context provider is much higher than the time required for processing them at the broker.

M2M scenario

M2M communications enable collecting crucial information from many objects via wireless network (e.g. short-range radio, 3G/4G network) to a backend server for aggregation and processing. It is widely used in many domains, like energy, health, security, transportations, remote maintenance, or Human-Computer Interaction (HCI). In logistics, exchanging contextual information (e.g. objects geospatial information, agents' health condition) between operational agents deployed in the field with company back office is very crucial. The monitoring of context changes enables real time supervision of processes execution and exceptions handling. As process execution may not comply with predefined plans, for instance traffic jams may delay a delivery plan, back office agents have to be alerted in case of emergency in order to be able to respond immediately (e.g. change plan or abort delivery).

In the following case study we use the cloud-based context management platform presented in section 4.1.4 for the implementation of a M2M scenario where a shipment truck has to deliver a product from a location (e.g. Paris) to another one (e.g. Caen). During this travel, contextual information of the truck are sent to the cloud platform (servers responsible for managing context in the cloud domain). Also, presence information of the back office agent are sent from the internal presence server (here we used Microsoft Lync Server) to the cloud domain hosting the context management. Context-awareness is implemented in the public domain thanks to the context composition language. It aims to send a voice message to the back office agent on its smartphone (where an Android supervision application is installed) if the temperature of the fridge exceeded a certain threshold and the agent is not available in front of the supervision interface (his presence status is offline).

The testbed environment is illustrated in figure 6.5. The GPS coordinates of the route between Paris and Caen was generated with CloudMade[1] Routing API, and played by a location provider java program. Temperatures of the truck fridge are simulated and published to the platform.

Figure 6.6 depicts the supervision interface that is Swing-based Graphical User Interface (GUI). It is a context consumer application that listen to changes in location of the supervised truck and temperature of its fridge, then display them in a suitable interface to help back office agent to track in real time delivery situation.

1. http://www.cloudmade.com/

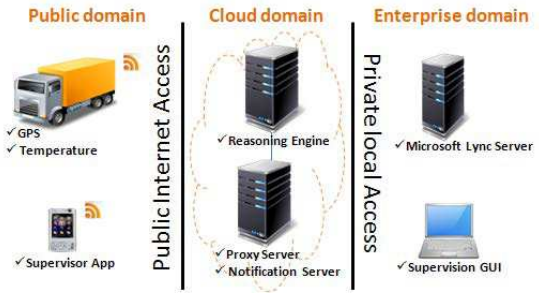

Figure 6.5: Test environment configuration

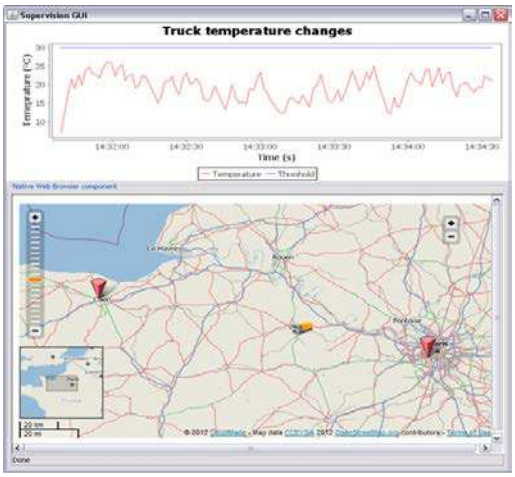

Figure 6.6: Supervision GUI

The performance evaluation is conducted in the following environment: a server running a Microsoft Windows Server 2003 SP2 with following hardware characteristics (Vender: Intel, Model: Xeon, CPU: 2.8GHZ, memory: 2 GB). On this server is installed a Jetty web server to run the proxy server and reasoning engine. Also, this server runs a CometD server to play the role of the notification server, to send context updates to subscribed clients. A context provider and consumer were developed as simple java program to publish context and receive updates. This program sends context publication messages to the context management platform. It runs on Microsoft Windows XP SP3 installed on Dell D630 laptop with the following characteristics (Vender: Intel, Model: Core 2 Duo, CPU: 2.2GHZ, memory: 2 GB). Detailed explanation of the role of each of these different components can be found in section 4.1.4.

Figure 6.7 depicts the response time of the context management platform

correspondingly to the number of the received context updates. The aim is to model this time as a function that takes as parameter the number of parallel requests in order to measure the reactivity of the framework and estimate future response time.

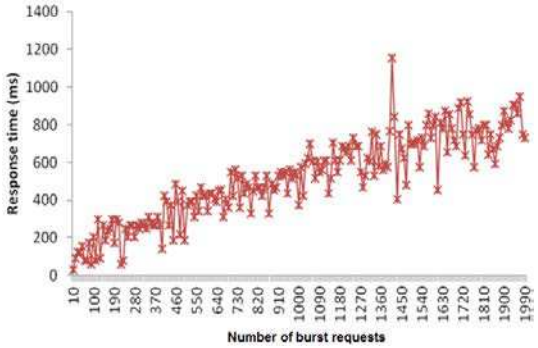

Figure 6.7: load performance

The variation of response time can be approximate with the following linear:

$$ResponseTime(RequestNumber) = \alpha + \beta * RequestNumber \tag{6.2}$$

Where the function parameters α and β can be approximately computed as follow:

$$\begin{aligned} \beta &= average\left(\frac{\Delta ResponseTime_{i+1,i}}{\Delta RequestNumber_{i+1,i}}\right) \\ \alpha &= average(ResponseTime_i - \beta * RequestNumber_i) \end{aligned} \tag{6.3}$$

After calculation we found $\alpha = 162.097766$ and $\beta = 0.35454545$. Response time increases slowly with the number of parallel request as the β ratio is less than 1. Performance test results show the platform ability to notify context information with a good response time (only 1 second for approximately 2000 parallel context updates and mostly due to network delay).

6.2 Application

The following case studies represent examples of innovative context-aware applications that perform environment sensing functionalities, maintaining a representation of the user's context, and adapting their behavior according to specific changes that may occur to the user context. The implementations demonstrate

the applicability of the proposed referential architecture of context-aware applications and the conceptual framework introduced earlier in Chapter 4.

6.2.1 Enhanced un-interruptibility

Advances in information and communication technologies), especially in the professional environments, are continuously enhancing the way how co-workers communicate and collaborate. In the same time, these technologies are increasingly adding an important amount of stress onto workers. In fact the diversity of the available communication channels (e.g., Email, IM, Video conferencing) lead to put the user in a complex situation as he/she has to choose between many options to reach someone else. Also, the user has to customize all the available channels to indicate his/her preferences regarding how he/she would prefer to be reached.

In addition, the diversity of the used communication tools may lead the user to lose the control on the way they can be reached and at what time. Causing an increase in the amount of notifications or interruptions caused by the reception of the communication request (e.g. when an email is received). The continuous stream of interruptions may cause degradation of the worker performance on his current activity or influence his/her choices of the future ones [87]. Hence, it's important to control when interruptions occur on behalf of the user in order to not affect his performance. The approach that can be considered to solve the interruptability issue is to delegate the control of interruptions to the user contacts by giving them access to the user contextual information that describe his/her situation. Such information will help users' contact to evaluate the importance, at a given time, of the interruption they will cause if they try to establish a communication with the user.

The enHancEd un-interruPtibility (HEP) system investigates a new form of multi-sensory composed data to increase the understanding of the user situation. Through HEP [88] we aim to reduce the overhead created by the management of the communication channels and the stress generated by the continuous interruptions on the user activity caused by answering incoming communications. This by delegating the user interruptions management to his/her contacts.

Usage scenario

We aim to tackle the interruptions resulting from receiving calls (or communication requests) at a non-appropriate time to avoid potential inconvenience . To be able to derive the desirable characteristics of any system that can provide answers to avoid interrupting the user current activity due to the reception of non-appropriate calls, we evaluate the usefulness of a communication man-

agement system under several scenarios. This evaluation is made in different environments both before and after the integration of this system.

Alice (hypothetical user in these scenarios) had a long working day with many phone calls, some meetings on the agenda, and an important amount of mails considered to be urgent. At a given time of the day, Bob, the other hypothetical user, calls Alice without having any prior knowledge of her availability or her work load.

Scenario 1: Using a featureless landline phone
Alice has on her desk a featureless landline phone. So the phone rings loudly on receiving calls, and she has to pick it up and hang it. But already the interruption has been instantaneous and complete. Alice is derailed from her train of thoughts, furthermore in case she was taking part of a meeting then everyone else in the same meeting will be annoyed and an aroma of social stigma is clearly in the air. Now, let us enable context-awareness capabilities and see the differences. This landline phone can only receive data from the corresponding utility pole or the enterprise private branch exchange (PBX) and knows nothing about the local environment. As a result, no real improvement can be achieved in this scenario.

Scenario 2: Using an availability management system
Alice is using an availability management system which is able to generate an accurate representation of her availability by analyzing her communication history and upcoming schedules. This information is then made available to anyone who is intending to call her, for instance by integrating an availability status to Alice contact information in the address book, in order to be aware of any potential incur that may result from calling her at the given time. In this scenario, Bob will have to check the availability of Alice before calling her and may eventually give up and try to call her later. But as there is nothing controlling the call establishment, there is nothing prevents him from making the call, thus Bob may end up disturbing Alice by initiating a new conversation.

Scenario 3: Using a communication management system
Alice is using a communication management system which is able to control incoming calls on her behalf. This system knows from her calendar schedule that she is supposed to be present at a meeting during those hours. So it decides to interrupt incoming calls or takes the caller to the voice message option based on an already defined rule. Later, when the meeting is over, Alice has no idea that there had been a call which may have been an important one or an emergency call. If the communication management system had not context-awareness capabilities, then Alice should check manually her call history or make a call herself within a certain time period in order to find out that she missed something very important. In this case, although the cost of interruption was mitigated, a po-

tentially important call is missed. While if the system is context-aware and can sense changes in Alice context and behave accordingly, then it may, for instance, have a rule that states to interrupt Alice to notify her about the occurrence of a call. As a result, Alice not only avoids undesirable interruptions during the meeting but she is also notified in an appropriate manner about any missed call.

Scenario 4: Combining availability and communication management
Now, Alice is using a system that combines communication and availability management. This system continuously senses Alice communication history as well as calendar information to keep updated her availability information. In addition, it's able to control incoming calls in order to reject them when they are received at a non-appropriate time. Now, when Bob tries to call Alice he will be promoted with her availability information. In case he ignores the fact that Alice is not currently available and initiates a communication, the system will handle this request and reject the call. When Alice becomes available, the system will notifies her about the missed call. In this scenario, not only availability information are passively displayed for callers but an active action is performed to prevent the occurrence of an interruption as a result of the incoming call. Furthermore, the system chooses the appropriate time to notify the user. In this case, we can clearly see the benefit of an intelligent communication manager.

Challenges

Communication management is a research domain that attracted many earlier works. Several research projects have already explored the use of contextual information for an enhanced accuracy of the context management. Many of them focused on the delivery of context information about users to any other one who may potentially try to call this user. For instance Cals.Calm [26] enables the sharing of user's information that describes his situation publicly on the service web page. This system relies on the user to manually edit the information about his current location and activity which may become a tedious and repetitive task. A more active approach is used in [89] where a scripting system is used to allow users to indicate their preferences regarding the actions (e.g. reject, accept, forward) that should be executed for handling incoming calls based on the call context (i.e. information about the caller, time, etc.). In the same philosophy, SECE [27] provides the user with a richer scripting language that enables the configuration of actions that should be executed under certain conditions. End-users are not supposed to be expert with any level of programming capabilities in order to be able to write complex configuration scripts and thus may find this kind of systems hard to use and customize.

In our case we are considering multi-channel interruptions management to the

contrast of other works who had focused on phone call as unique communication channel. Challenge: integration with legacy systems.

The challenge of an interruption management system is to provide an accurate and personalized representation of the user availability, which goes beyond the simple sharing of low level information (upon their availability) to describe the user situation. It should be based on the combination of a multitude of context information with the consideration of historical data. The user must maintain its control on the system to customize the system operation rules and outcomes; otherwise the system would be hardly usable. As the system aims to reduce the user overhead, its interaction with the user should be the less possible as the system will not be interesting to use if it generates as much interruptions as those generated by incoming communication requests. Furthermore, the system should not over aggregate the generated information about user availability as the latter depends highly on the corresponding communication channel. For instance, the user may be available for instant messaging communications but not available for phone communication because he/she is in a meeting. Similarly, the user might be available for phone conversation because he is walking while not be available for instant messaging or text conversations.

The adaptation of an interruption management system has to maintain in an intelligent way a tailored representation of the user availability. The subsequent sections describe the instantiation of the conceptual framework of context-aware applications and the use of the underlying adaptation mechanisms for the specific installation of the HEP system.

Software architecture

HEP is an attempt to make use of the inherent relation between interactivity of the user with communication requests and his/her availability for handling new incoming requests. The HEP system is integrated with the user communication tools to monitor the user everyday interactions with the different available communication channels. It keeps track of read/unread emails, number of answered/unanswered phone calls as well as composed calls, and the amount of time spent in meetings. It generates then a score representing the user availability that it tries to keep up-to-date continuously. This score is shared with the user's contact just like how his/her presence information (i.e. online, offline, away, busy) are shared via any commercial instant messaging solution.

For the end-user, the HEP system mainly consists of the plugin presented as an extension of the user favorite communication software (herein Outlook). This plugin extends and enriches the interface with an additional set of information about the user contacts' availability. The following figure 6.8 depicts the software architecture that represents the different parts of the HEP system.

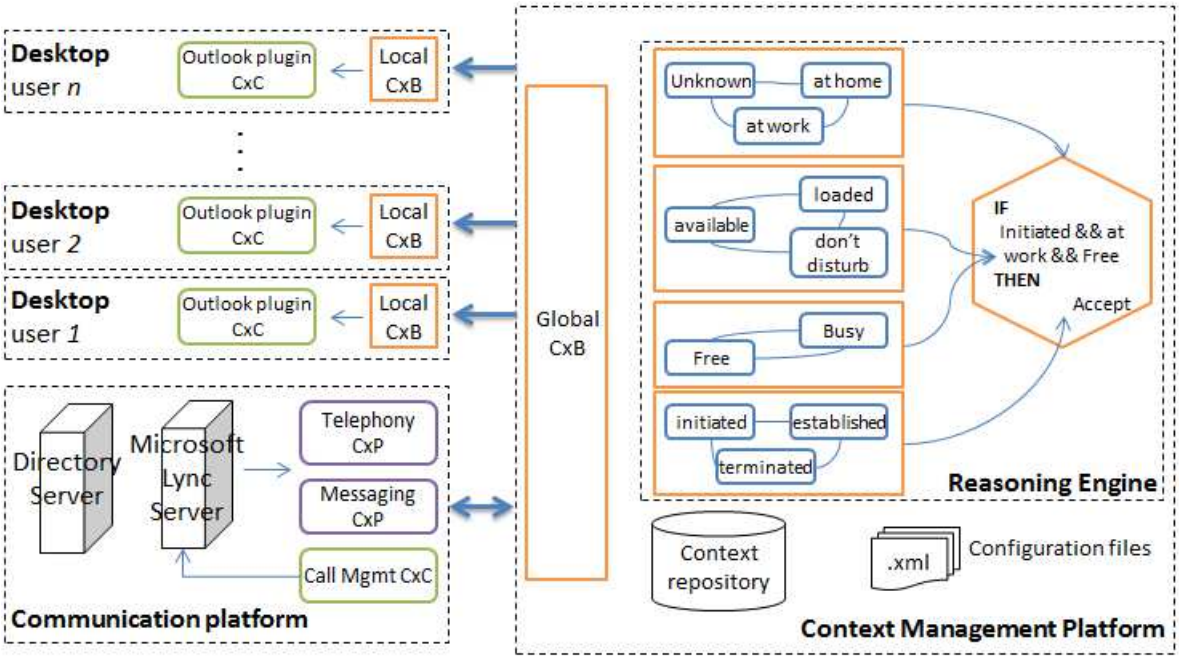

Figure 6.8: Architecture overview of HEP

The different elements composing the architecture (as depicted in the figure) are: A PC client responsible for retrieving raw data from virtual sensors placed on Microsoft communication suite (Email, Calendar, IM, and Telephony); computing user status for communication mean and interacting with the context management platform. An Outlook plug-in that provides the user interface, it enables the user to set his preferences (e.g. the status that should correspond to a given load level), and furthermore it enables the user to see the statuses of each of his Outlook contacts. The Context management platform provides storage and management functions. It enables the PC clients to subscribe and publish their status, and enables the Outlook plug-ins to request the status of other users. An administration interface is available to set global rules for status computation.

Computation of user status is based on information about the history of usage of communication services (e.g. email) and desktop applications (e.g. Word, Excel, PowerPoint). The frequency of computation and freshness of status depends on the related communication service, but can be fixed by users when they specify their preferences.

The figure 6.9 provides an overview of the instantiation of the generic architecture for context-aware applications proposed in chapter 4 and further detailed in chapter 5 for the specific case of the HEP system. This figure illustrates the

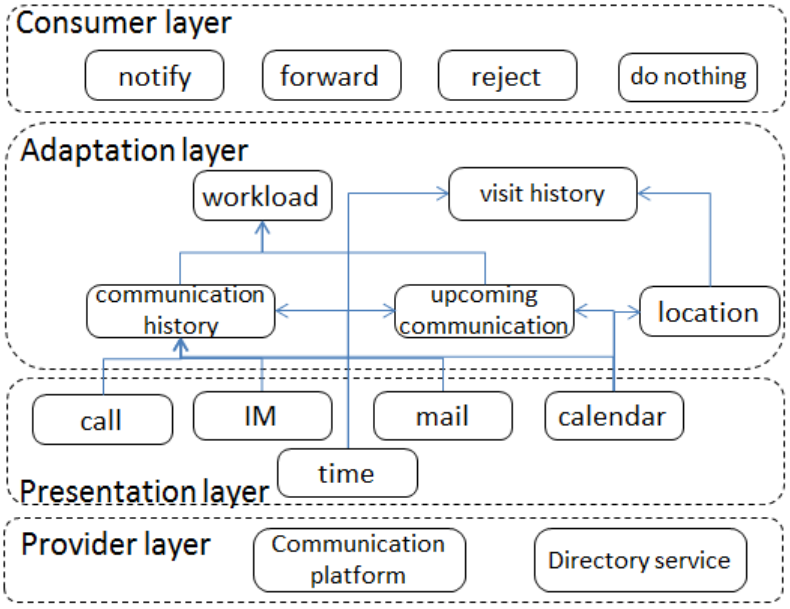

Figure 6.9: HEP instantiation of the generic architecture

distribution of the core components of the application through the different layers of the architecture. The following subsections describe in details the component at each layer.

The underlying communication platform, at the provider layer, is based on Microsoft Lync Communication Server. The provider component deployed on top of this platform is a web service uses the UCMA, provided by Microsoft Lync, to subscribe the communication events of each user. It then uses the API of the context management platform to feed the whole system with a stream of events related to the usage of communication services (phone, IM, e-mail, calendar).

The information circulated within these communication events are centralized in the context management platform to be available for the different instances of the HEP application. At the presentation layer, the acquired information are modeled into one of the following classes: call, IM, mail and calendar based on the corresponding communication service referred in the event. In addition, time information are modeled to be able to manage communication history and to order communication events based on their instant of occurrence.

At the adaptation layer, the stream of communication events is processed in order to refine the context model that represents the knowledge about the user status, emotions, activities and workload. Then, at the consumer layer, this knowledge can be shared with the user contacts so that they can be aware of his situation when they try to contact him or it can be used to control incoming

conversation on behalf of the user.

Context Modeling

Based on the information provided by the communication platform (as illustrated in the previous figure), the presentation layer maintains the complete view of the user context that includes information directly acquired like the state of the call: whether it was answered or rejected, etc. Here is a detailed description of the list of information available from the provider layer and used to model the user's context:

- Time: holds time information in milliseconds.
- Call: holds information about a corresponding call conversation, if it was a received or an emitted call and how it was handled: rejected/answered/missed.
- IM: holds information about a corresponding instant messaging conversation.
- Mail: holds information about a corresponding mail conversation.
- Calendar: holds information about a scheduled meeting and its location.

The different contextual information (both sensed and deduced ones) are gathered in an UML data model as illustrated in Figure 6.10. The Context class is the root class in the model that is common to any type of context information. The Communication class gathers the shared attributes of interactive communication tools, while information specific to a communication tool is gathered in a specific class (e.g. IM, email, phone). The calendar is considered a communication tool as it holds information about meetings which are supposed to gather two or more people to discuss a common concern.

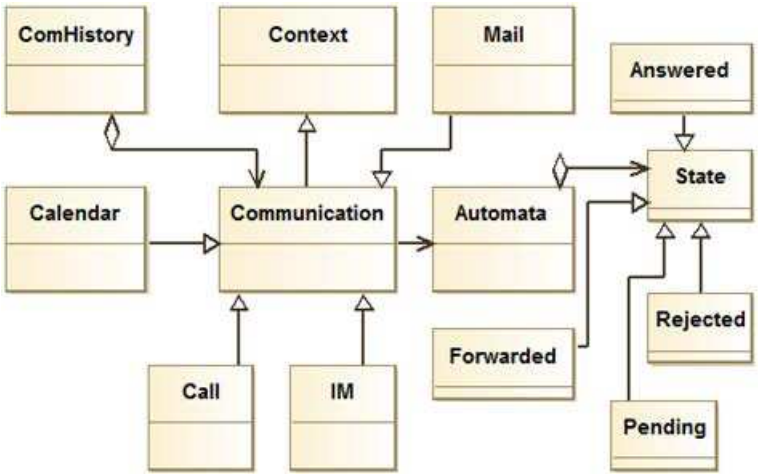

Figure 6.10: HEP Data Model

The generated workload information is hold in the "Load" attribute of the Communication class. The value of this attribute is deduced from workload information of different available communication channel (IM, mail) represented by the sub-classes of the Communication class. For instance, the workload corresponding to the calendar is the ration of the total amount of meeting time to work time. Some communication tools provide presence information, they are represented in the UML diagram thanks to the "Available" which represents whether or not the user is available at a given instant in a communication tool. For instance, the calendar availability corresponds to whether the user is currently in a meeting or not. The timestamp represents time information used to keep track of the moment of occurrence of an event (e.g. incoming communication request). Another time variable is validity which represents for how long the sensed information remain valid. The value of the later depends on the communication tool, for instance it is 5mn for Mail, 15mn for Calendar. In addition, the model holds more dynamic information about a communication; for instance:

- The ratio of missed requests (e.g. missed phone calls, IM requests or unread mails) to the received one;
- The ratio of engaged communication to the received ones;
- The ratio between free time and total amount of meeting that represents the overall availability;
- The ratio of unread message from the user's voicemail to the stored ones.

Context processing

The context processing function aims to process the raw information acquired by low layers to generate more meaningful knowledge which will be maintained in objects instantiating the classes of the precedent UML diagram. Examples of the resulting information include:

- Location: holds information that corresponds to the location declared as an attribute of a scheduled meeting.
- Communication history: holds a list of all communication till now which were made by the user.
- Upcoming communication: holds a list of all upcoming scheduled meeting which are declared in the user calendar.
- Visit history: holds a list of all location where the user has been for an appointment.
- Workload: indicates the workload level information which is generated as a result of processing and aggregating previous information.

The generation of these information is straightforward, for instance the communication history is created by collecting all instances of the user conversations (calls, IM, emails) and ordering them based on their occurrence time.

However, the context information representing the user workload requires a more complex derivation process that depends on the communication tool (e.g., IM, mail, phone, calendar) and is described as follow. The sensed information are used to compute the work load of a user on a given communication tool in order to determine the user status and whether or not he can accept incoming requests on this communication tool (figure 6.11) . We defined rules for calculating the work load level for each communication mean (IM, mail, phone, calendar).

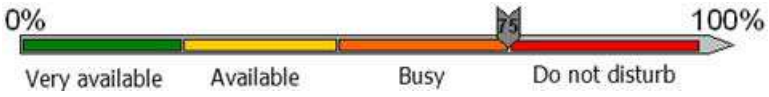

Figure 6.11: User status based on his work load level

In case of the calendar, the related workload can be described as follow: if the user is currently in a meeting then we set his calendar work load to 100%. Then, the more the meeting start time gets closer, the more the calendar work load goes higher (e.g. 5m before a meeting, work load reaches 75% and user status become busy'). If the user is not in a meeting then calendar work load is the ratio of meeting duration in the rest of the day to the remaining work time. Here is an example of the calculation of the calendar work load for a given user at different time of the day. We consider that a work day start at 8:00 and finish at 18:00, and the user have a first meeting from 9:00 to 11:00 (2h duration), then a second one from 15:00 to 18:00 (3h duration). Thus, the workload will change during office hours as follow:

- At 8:00 the work load is (2+3)/10 = 50%, i.e. sum up the total duration of time during which the user is in meeting and divide by the total amount of time available,
- From 9:00 to 11:00 work load is 100% (user in meeting),
- At 12:00 work load is 3/6 = 50%, i.e. sum up the total duration of time during which the user is in meeting and divide by the remaining amount of time,
- At 14:00 work load is 3/4 = 75%, as described in the precedent case,
- And between 15:00 and 18:00 work load is 100%.

Similar procedure is applied to the other communication tools and the different workload statuses are then used to compute a global status that reflects the user overall workload. For computing the later, predefined weightings that can be modified by the end-user are attributed to the workload of each communication means then the different values are summed.

The level of workload of a user on a given communication service (e.g. agenda, email, instant messaging, or phone) is represented to the end user as an avail-

ability status. Example of the different statuses of a user are depicted in Figure 6.12: "Very available" corresponds to the state where user is highly available for receiving communication requests (e.g. phone call, IM request); "Available" corresponds to the state where user can receive call requests; "Busy" corresponds to the state where user can weakly respond to a call request; "Do not disturb" corresponds to the state where user cannot respond and will potential refuse incoming communication requests.

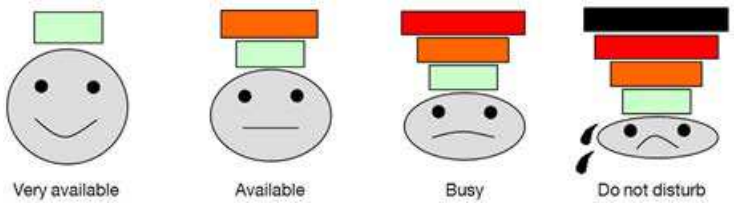

Figure 6.12: The different HEP statues

Implementation

The HEP system can be used as a context-aware system for recommending communication means for enterprise employees. The client side of the HEP system that enables the user contact to visualize availability information are integrated directly to the commonly used communication software to make easier for colleagues to see each other workload status. Such information can be used by the caller to decide if he can interrupt the callee, and whether is it better to use a communication service (e.g. email) than another service (e.g. phone) in order to reach the callee. For instance, let suppose that Alice wants to call Bob for an urgent matter. Bob is at this moment in a conference call, but he is still reading his emails and answering them. With HEP, Alice will see that Bob is busy on the phone, but available by email. Thus, she decides to send him an email instead of calling him, although her demand is urgent. The next figure 6.13 illustrates the integration of HEP with the enterprise Directory service, so that when users are looking for all possible ways to contact someone they can find along his up-to-date availability information.

In addition, a Microsoft Outlook plugin (as depicted in the following figure 6.14) is provided to give an instant access to contacts availability without leaving the main client's graphical user interface.

The diversity of these clients represented a real challenge for the application realization as any single modification of the application internal structure may lead to a modification in the different clients. The use of a RESTful architecture

Figure 6.13: Integrating HEP with the Intranet

with well-defined interfaces to the platform (and as a result of the application) limited the complexity of the clients development as the need of modification is required only in the case of the format of the transported data changed.

6.2.2 Enterprise Social Graph

The enterprise environment gathers different kind of computing devices: servers forming the IT infrastructure where are hosted the enterprise services (e.g. mail, directory) and a collection of end-user devices that range from PC, laptops to smartphones. These devices constitute a promising source of information on the activity of the users. However their integration and exploitation represent a challenge as these information have an extremely heterogeneous nature: they don't have similar validity time, may not update regularly, may represent redundancies, each source expose a different interface to third-party components, etc. These challenges require a flexible way to model context that can evolve over time by adding new set of information or removing no longer need information. Also, the way the reasoning procedure needs to consider the timeliness nature of the information itself in order to return valid results.

In this section, we show how the context management platform was used to connect the two different worlds of the enterprise infrastructure and the end-user devices. The platform did not only providing a flexible modeling approach

Figure 6.14: Integrating HEP with Outlook

through the use of the context graph but also provided an event driven reasoning that support real time reactions. In addition, the platform enabled a seamless integration of new sources or consumer of information as well as a considerable support for the architecture scalability.

Software architecture

Enterprise Social Graph (ESD) [50] is a social network application that provides a social dimension to the Enterprise Directory by extending the static links between employees (e.g., hierarchical relations) with links maintained dynamically based on users' activity. It relies on the following context providers: an Lightweight Directory Access Protocol (LDAP) client to provide employees profile information and managerial relations, and a Microsoft Lync client to provide information related to communications. In addition, it uses a provider connected to Plaza an internal enterprise social network. The provided information are used to maintain a social context graph that will be explored by the application (i.e. CxC). For example, communication context is used to evaluate the proximity score of two employees in terms of the amount of communications established through the Lync platform. This application is a kind of search engine that allows user to find people which are related to a certain set of keywords and also related to the user search history. Also, it provides a way for a step by step navigation by following links starting from a given entity to reach next entities. The following figure 6.15 depicts the software architecture that represents the different parts of the ESD system.

The different components composing the architecture (as depicted in the figure 6.15) are: The front-end of the ESD application is responsible for the in-

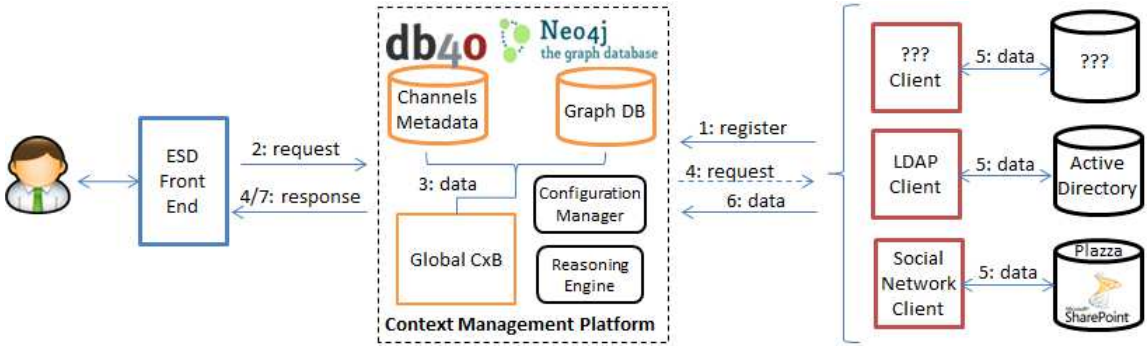

Figure 6.15: Architecture overview of ESD

teraction with the end-user through a web interface, as well as exposing the application functionalities to enable the user entering his/her search queries and visualizing the search result. At the back-end side, a set of information sources are deployed to collect raw data from virtual sensors placed on Microsoft Active Directory server (i.e. LDAP server), Microsoft SharePoint server. Between the two environments, the context management platform is deployed on a dedicated server to integrate the different collected information and to provide data storage/management capabilities. It handles the generation of more meaningful information and exposes data access functionalities to frontend applications.

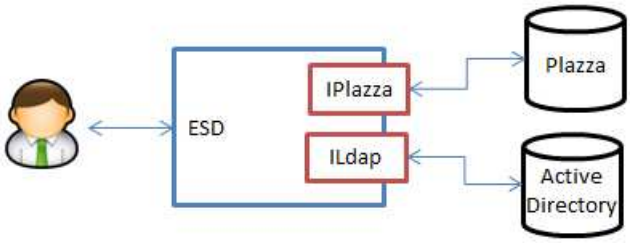

Figure 6.16: Alternative monolithic architecture for ESD

The benefit of using the context management platform over a standalone application based on a monolithic architecutre (as depicted in figure 6.16) is twofold: first it enables the integration of additional information sources with the minimum possible effort; second it enables a seamless replacement of an existing source with another one without having to modify any part (e.g. the data wrapper) of the application. In the first case, the effort consists of the modification of the XML configuration that indicate to the platform how to create new type of elements of the graph (i.e. nodes and edges) and their integration

to the existing graph. In the second case, the only modification needed is the replacement (in the XML configuration) of the new source address which will be used by the platform to request information relative to the managed graph elements (i.e. nodes and edges).

Context Modeling

The combination of information provided by both the communication platform and directory infrastructure enables the presentation layer to maintain a complete view of the employees' profile and contextual information. Some of these information are directly acquired like profile information (e.g. name), some relatively static information (e.g. function) as well as information describing the relations between employees (e.g. hierarchical relationship) or between an employee and its work environment (e.g. department membership). Following is a detailed description of a subset of the list of information available from the provider layer and used both for user modeling and modeling the user's context:

User holds profile information concerning a given employee like name, function, etc.

Communication represents a relation between two users, it holds information about a corresponding conversation (whether a call or instant messaging conversation), like duration or the nature of the communication received or an emitted, the corresponding state (i.e. rejected, answered, missed).

Contact holds information about a relation between two users when one of them has the other on his address book.

Hierarchy holds information about a one direction connection between two users representing a hierarchical relation.

The different contextual information (both sensed and deduced ones) are gathered in an UML data model as illustrated in the following figure 6.17. The User class represents the main class for gathering information that concerns a given employee. The Context class is the top-level class in the model for representing contextual information. From it is derived a set of classes to hold information about different kind of relationship that concern a user.

Context processing

The context processing procedure aims to process basic information directly acquired by lower layers in order to generate more interesting information. The resulting information are then used to update the class instances of the previously introduced data model. For instance the Contact relation between two employees is regularly updated with dynamically generated information representing the strength of the relation. Examples of the set of information used by the processing

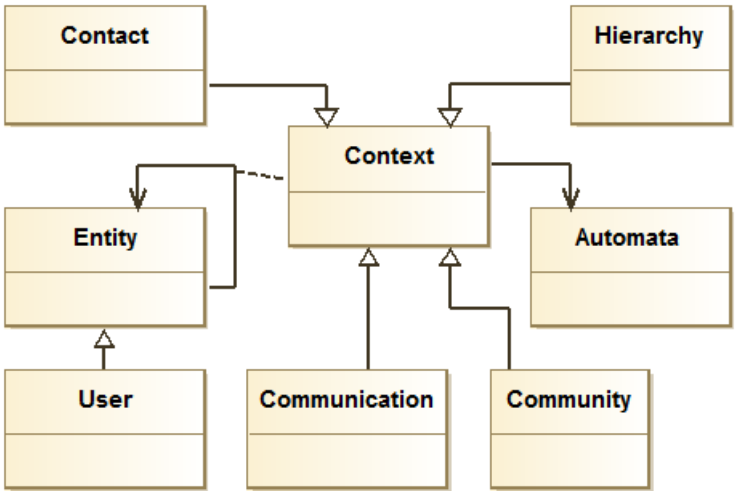

Figure 6.17: ESD Data Model

procedure include: the communication history that represents the communication made by a given user, the contact list that represents the list of the declared contacts for a given user, the communities list of which a given user is member. Other information can also be used to calculate the strength of a relation, for instance information about common locations (e.g. work place, where meetings were held) which were attended by two employees. The generation of the strength information of a relation between two users is illustrated in Algorithm 6.2.2; it is based on calculating a proximity index based on the number of shared contacts and communities between these users. The proximity index depends also on the number of non-shared contacts/communities of each one. We can also consider other information like number of conversations established whether successfully established, rejected or forwarded.

The procedure for retrieving the contacts of a user or communicates the user is member of relies on the graph traversal interface which is provided by the context management platform. The following figure 6.18 illustrates how the platform handles graph traversal requests: first, it checks if the starting node is available in the local graph database, if not then it requests this node information from the corresponding provider to update the database by creating the node and its outgoing edges. If the first node information are available, it checks the out edges that satisfy the traversal conditions and then their validity: if the edge is valid it will be taken otherwise an update request is sent to the corresponding information provider. In case, the edge validity time is not expired then it is added to the result list. After, exploring the resulting outgoing edges of the current node, the landing nodes are then explored until all commands in the traversal query

Algorithm 1 Calculating person to person proximity

Function
Contacts(p) returns the list of contacts for the corresponding person p
Communities(p) returns the list of communities that person p is member of
Input:
Number of current user contacts N,
Contacts of current user $CN = \{cn_1, \ldots, cn_n\}$,
Number of current user communities M,
Communities on which current user is member of $CM = \{cm_1, \ldots, cm_m\}$,
Output:
Proximities $PR = \{pr_1, \ldots, pr_n\}$ between current user and each one of his contact

```
1:  for i from 1 to N
2:    pr_i ← 1 / N
3:    count ← 0
4:    for each contact c_j ∈ Contacts(cn_i) do
5:      if c_j ∈ CN then
6:        count ← count + 1
7:      end if
8:    end for
9:    pr_i ← pr_i + count / N
10:   count ← 0
11:   for each community cm_k ∈ Communities(cn_i) do
12:     if cm_k ∈ CM then
13:       count ← count + 1
14:     end if
15:   end for
16:   pr_i ← pr_i + count / M
17: end for
18: return PR
```

are processed. At this moment, the resulting information are formatted to be returned back to the component that emitted the traversal request.

Implementation

The client side of the ESD application allows the user to search for new contacts by submitting search keywords. The result as illustrated in figure 6.19 is ordered and numbered (as depicted in top left of the contact card) based on the proximity indices of the returned users. A low numbered contact corresponds to a closer in proximity to the initial user who submitted the search query. Once the result for the current search query is visualized, the user can select one of the resulting items to explore the subsequent items closely related to this item.

Figure 6.19 depicts the presentation of the application search result. The scroll bar at the bottom side is used to control the depth of the visualized result, low levels (left) reflect higher relevance. The little hairspring in the bottom left of the interface, represent the previous search result. They allow users to see how

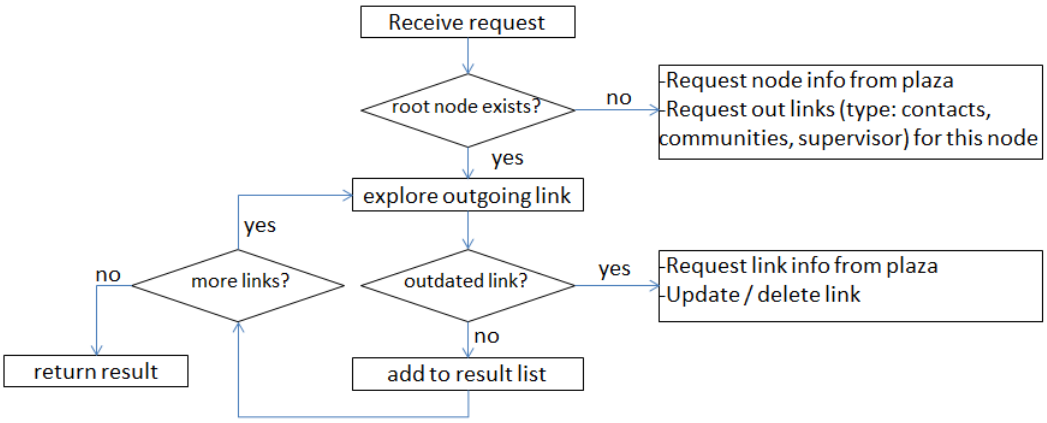

Figure 6.18: Handling graph traversal requests for information about a specific user

he reached the current search result and be able to return back to previous result. This interface also allows visualizing profile information of a given person.

The ESD application was tested with information corresponding to a group of our teammates at Orange Labs. Figure 6.20 depicts a subset of nodes and edges from the context graph on which the ESD application relies. This subset illustrates the diversity of managed data as it contains nodes representing employees (e.g. Marc M.), discussion communities (e.g. HTML5) or location (e.g. offices) and as a result the variety of relationships.

The feedbacks received from the early adopters were very positive especially concerning the quality of the result returned by the application and for the intuitiveness of its interface. This case study shows the benefit of modeling context information as a graph and the use of graph traversal algorithm to reason on context. However, it shows also the important volume of data that needs to be managed to maintain a context graph.

6.3 Summary

This chapter discussed the application of the concepts proposed by the thesis and the use of the development kit provided through the context management platform in the creation of some context-aware applications. The successful implementation of the different case studies demonstrates the benefit of the proposals in terms of support for all aspects regarding the context-aware applications (i.e. context management and processing). It also shows the applicability of the concepts for a wide range of application and especially for applications in the

Figure 6.19: A screenshot from the ESD interface

enterprise domain.

The realization of the different case studies presented in this chapter provides a useful way to experiment with the engineering of context-aware applications based on the proposed context management platform. In addition, along the realization non-anticipated challenges were faced and from which new ideas raised creating new opportunities for future works. This section elaborates on the general lessons learned from this experience gained in the application of the thesis contribution.

The decomposition of the context-aware application into several functional components with a set of well-defined interfaces between the different components implementing these functions allows a modular implementation that significantly simplifies the representation of the application. More generally, this helps support the reusability of each part of the application, the definition of the interfaces of each part in a rigorous way and supporting the scalability of the application. In addition, it helps greatly in achieving a certain transparency level thus making the maintenance and debugging of the applications much easier.

The XML configuration files that specifies the connection between the different parts of an application represents a program in itself writing in the context management platform language and vocabulary. This language represents a high-

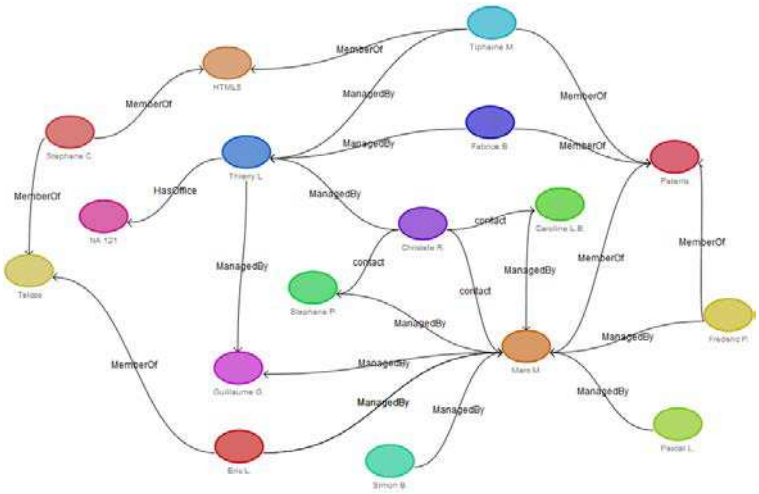

Figure 6.20: A subset of the Context Graph of ESD

level programming paradigm that hides and abstracts complex operations from the developer and thus easing the creation and modification of the application behavior. However, it grows in size and complexity as the context processing procedure defined for this application gets more complex. Therefore, dealing with complex XML files may become a tedious task and a tool for an automatic generation of these file will be of great help for developers. An opportunity would to be to investigate the Graphical Editing Framework (GEF) which is an enabler technology provided by the Eclipse project [1] for creating rich graphical editors for DSL (Domain Specific Language).

The client part of the context-aware application is the component responsible for providing users an access to interact with the application. It should provide an intuitive interface that empowers the user while hiding the complexity of the application. Unfortunately, the realization of the different case studies allowed us to realize that the client is also the most complex component to design as it depends on the nature application itself and on how it will presented to the user (e.g. mobile, web, or desktop client). The client development needs different technologies that cannot be easily unified or integrated, resulting in a limited possibility for reusability or knowledge transfer concerning the how-to of the client implementation. Cross-platform tools may be a solution as they provide code-once-deploy-everywhere capabilities making it possible to maintain one single code base and being able to generate from it many client instances one for each specific platform.

1. http://www.eclipse.org

Understanding the application domain is very important and may be challenging in some cases. This importance rises from the domain-dependency of some tasks of the application, for instance the context modeling and processing. However, some of the core components are domain-independent and can be shared across applications of different domains. For instance, the different scenarios rely on the communication context provider which is an abstracted components to capture contextual information on the user communications.

Chapter 7

Conclusion

7.1 Summary

Facilitating the engineering of effective context-aware applications is a challenging task due to the complex operations related to the management of context as illustrated in section 2.2.2. Context-aware applications potentially bring usability problems as the end-user lacks visibility on how the context-aware behavior of the application is determined or on how the application consumes his/her contextual information. Furthermore, there is no unique definition for context nor for its representation within context-aware applications as illustrated in section 2.2.1. Thus not only, should the developers be empowered by providing flexible tools that help developing and debugging context-aware applications, but also other stakeholders should be empowered for the definition of the application context-aware behavior.

The investigation of the related works in terms of approaches facilitating the management and integration of context data within applications revealed that the current proposals hardly provide a flexible support. The context representation approaches focus primarily on entities and represent contextual information as attributes for this entity. Such a representation lacks flexibility and fails to express connections between entities. The tools and frameworks providing support for engineering context-aware applications tend to use non-flexible context representation and hardly customizable context processing mechanisms.

In addition, the implementation of the context-aware behavior is usually influenced by the developer perspective. And developers cannot anticipate the different needs of the users under different situation where the application will be used. Thus, some aspect of the user perspective may be overlooked leading to usability problems. The user can discard an application because he do not understand its behavior.

The main contributions of thesis are the proposals of a context-centric representation that enables the use of context data to express connections between entities and creates a context graph that facilitate the exploration of entities sharing similar context. In addition, the thesis proposes a new conceptual framework to facilitate the engineering of context-aware applications. It provides a multi-level support (e.g. at the programming or deployment levels) and a holistic view covering all application concepts. This is the result of decomposing context-aware applications into several functional components and explicitly describing interfaces to handle the dependencies among them. Also, the architecture of the framework is universally applicable for engineering applications in different domains.

The thesis contributions are validated through the implementation of different case studies. These case studies demonstrate the successful use of the proposed conceptual framework and how the different features were used to facilitate the implementation of each part.

A transfer to the industry of some concepts proposed in this thesis was conjointly carried with a team of engineers from the Orange Labs (the R&D department from the telecom operator Orange formerly France Telecom). A more robust implementation of the proposed context management platform has been achieved by Orange with a goal of building a global context enabler that will be used in the Orange Labs projects. The flexibility of the platform and its adaptability to very various needs have been notably appreciated by Orange.

7.2 Future work

The identified research directions to extend the contributions described in this thesis are as multiple. The context management platform described in the thesis can be extended further to facilitate the developer task by providing more useful tools that can be used in a daily bases and integrated to the development environment. For instance, the edition and debugging of reasoning configuration files can be made easier if a dedicated graphical editor or plugin is also provided. In addition with the current implementation, new deployed applications on the platform cannot automatically discover already existing components (e.g. context providers), only a manual browsing of the list of available components is possible. As components can currently publish their features and capabilities, it would be interesting to add to the platform a search feature for looking for components having specific capabilities.

As the developer may lack an overall vision on the execution environment of the context-aware application, it is important to capture feedback (explicitly or

implicitly) from the user. This feedback may describe the user quality of experience and used as an input to the adaptation process in order to adjust or correct the context-aware behavior and better match the user need. Furthermore, the user feedback can be used at the context source level for instance to select the right value of a contextual information when the available information have a very low quality. The context management platform described in this thesis support the representation and the use of quality information across the different layers of the adaptation process. However, the way an application request the user feedback or decide when the feedback will be useful depends on the application and its interface with the user, thus it is not easy to provide a universal mechanisms for engaging the users. It will be interesting to further investigate the use of feedback in context-aware applications through different case studies. The implementations will help evaluating the support provided by the context management platform and detecting limitations.

Furthermore, the context management platform provides an advanced support for the engineering of context-aware applications at every stage of the application lifecycle (i.e. design, development, deployment). It empowers the developer through the abstraction of common mechanisms related to management of context (e.g. acquisition, modeling, processing) and adaptation (e.g. event triggering). The capabilities of the platform are hidden and not accessible directly to the user, there is no direct interface toward end-users for helping them controlling or customizing the context-aware behavior of the application. It will be interesting to further investigate common ways to help users (e.g. modifying application decisions) of context-aware applications in order to avoid usability problems that may make the application useless. This is a challenging problem as the empowerment depends highly on the nature of the interface provided to the user to interact with the application (e.g. gesture-based) and to receive feedback from it (e.g. screen display).

Orange as a telecom operator owns context information coming from the network infrastructure (e.g., user location, current network load, etc.) and from users' mobile device due to its relationship with terminal manufacturer (e.g., Samsung). Furthermore, Orange has a direct billing relationship with end-users, and owns simple profile information and communication statistics about subscribers. Thus, it can act as a context broker and extends its business model to build a context-aware ecosystem by providing controlled access to this information to 3rd party service providers through API. The broker-based architecture of the context management platform proposed in the thesis fits naturally with the role of a telecom operator, however further investigations (e.g. on scalability, performance) are needed in order to meet requirements at the Telco scale. Work is ongoing at Orange on these topics.

References

[1] Y. Benazzouz, *Context discovery for the automatic adaptation of services in ambient intelligence*. PhD thesis, 2011. xiii, 61, 71

[2] A. Tugui, "Calm technologies in a multimedia world," *Ubiquity*, March 2004. 12

[3] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal Communications*, vol. 8, pp. 10–17, 2001. 13

[4] S. Lee, S. Park, and S. Lee, "A study on issues in context-aware systems based on a survey and service scenarios," in *Proceedings of the 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing (SNPD '09)*, pp. 8–13, 2009. 13

[5] T. Haaker, B. Kijl, L. Galli, U. Killstrm, O. Immonen, and M. D. Reuver, "Challenges in designing viable business models for context-aware mobile services," in *Proceedings of the 3rd International CICT Conference, Mobile and Wireless Content, Services and Networks*, (Technical University of Denmark, Center for Information and Communication Technologies, Kongens Lyngby, Denmark), 2006. 13, 14

[6] A. Bouabdallah, F. Toutain, M. Szczerbak, and J.-M. Bonnin, "On the benefits of a network-centric implementation for context-aware telecom services," in *In Proceedings of the 15th International Conference on Intelligence in Next Generation Networks (ICIN11)*, (Berlin, Germany), October 2011. 14

[7] A. Dey and G. Abowd, "Towards a better understanding of context and context-awareness," in *Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, ACM Press*, (New York, USA), 2000. 15, 61, 70

[8] K. Henricksen, *A Framework for Context-Aware Pervasive Computing Applications*. PhD thesis, The University of Queensland, Australia, 2003. 15

[9] Z. Zhenzhen, N. Laga, and N. Crespi, "User-centric service selection, integration and management through daily events," in *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM'11)*, (Seattle, USA), March 2011. 15

[10] P. Mehra, "Context-aware computing: Beyond search and location-based services," *IEEE Internet Computing*, vol. 16, March-April 2012. 16

[11] J. Simoes, P. Weik, and T. Magedanz, *The Human side of the Future Internet*. Towards the Future Internet Emerging Trends from European Research, IOS Press, 2010. 16

[12] A. T. Schreiber, H. Akkermans, A. Anjewierden, R. Dehoog, N. Shadbolt, W. Vandevelde, and B. Wielinga, "Knowledge engineering and management: the commonkads methodology," *Cambridge, MA: The MIT Press*, vol. 1, 2000. 19

[13] M. Szczerbak, F. Toutain, A. Bouabdallah, and J. Bonnin, "Collaborative context experience in a phonebook," in *The First International Workshop on inter-Clouds and Collective Intelligence (iCCI-2012), The 26th IEEE International Conference on Advanced Information Networking and Applications (AINA-2012)*, (Fukuoka, Japan), March 26-29 2012. 20

[14] O. Kwon, K. Yoo, and E. Suh, "ubies: Applying ubiquitous computing technologies to an expert system for context-aware proactive services," *Electronic Commerce Research and Applications*, vol. 5, no. 3, pp. 209–219, 2006. 20

[15] P. Coppola, V. D. Mea, L. D. Gaspero, R. Lomuscio, D. Mischis, S. Mizzaro, E. Nazzi, I. Scagnetto, and L. Vassena, "Ai techniques in a context-aware ubiquitous environment," *Pervasive Computing, Computer Communications and Networks*, pp. 157–180, 2010. 20

[16] M. J. OConnor and A. Das, "Sqwrl: A query language for owl," in *In Proc. Of 6th OWL: Experiences and Directions Workshop*, 2009. 21

[17] P. Ziafati, F. Mastrogiovanni, and A. Sgorbissa, "Fast prototyping and deployment of context-aware smart outdoor environments," in *Seventh International Conference on Intelligent Environments*, 2011. 21

[18] C. A. U.-B. M. L. for Model-Driven Development of Context-Aware Web Services, "Q.z. sheng and b. benatallah," in *Proceedings of the International Conference on Mobile Business (ICMB'05)*, 2005. 21

[19] M. Wieland, D. Nicklas, and F. Leymann, "Context model for representation of business process management artifacts," in *International Conference on Economics and Business Information (IPEDR'11)*, vol. 9, pp. 46–51, 2011. 22

[20] M. Knappmeyer, S. Kiani, C. Fra, B. Moltchanov, and N. Baker, "Contextml: A light-weight context representation and context management schema," in *5th IEEE International Symposium on Wireless Pervasive Computing (ISWPC)*, (Modena, Italy), 2010. 22

[21] N. Koch and M. Wirsing, "The munich reference model for adaptive hypermedia applications," in *In Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'02)*, (London, UK), 2002. 22, 23

[22] A. Gupta, C. Forgy, A. Newell, and R. Wedig, "Parallel algorithms and architectures for rule-based systems," in *In Proceedings of the 13th annual international symposium on Computer architecture (ISCA '86), IEEE Computer Society Press*, (Los Alamitos, CA, USA), 1986. 22

[23] E. Knutov, P. D. Bra, and M. Pechenizkiy, "Generic adaptation process," in *Proceedings of the WABBWUAS'2010 Workshop on Architectures and Building Blocks of Web-based User-Adaptive Systems*, pp. 13–24, 2010. 23

[24] R. Baloch and N. Crespi, "Addressing context dependency using profile context in overlay networks," in *In proceedings of the 7th IEEE Consumer Communications and Networking Conference (CCNC'10)*, (Las Vegas, NV, USA), January 2010. 24

[25] D. Melinger, K. Bonna, M. Sharon, and M. SantRam, "Socialight: A mobile social networking system," in *Poster Proceedings of the 6th International Conference on Ubiquitous Computing*, (Nottingham, England), 2004. xx, 25, 28, 29

[26] E. R. Pedersen, "Calls.calm: enabling caller and callee to collaborate," in *extended abstract on Human factors in computing systems (CHI01)*, (New York, USA), pp. 235–236, 2001. xx, 25, 27, 120

[27] O. Boyaci, V. Beltran, and H. Schulzrinne, "Bridging communications and the physical world," *The IEEE Internet Computing*, vol. 16, pp. 35–43, March-April 2012. xx, 25, 62, 120

[28] onx, "automate your life," 2012. `https://www.onx.ms/`. xx, 25, 62

[29] M. van Setten, S. Pokraev, and J. Koolwaaij, "Context-aware recommendations in the mobile tourist application compass," *Adaptive Hypermedia and Adaptive Web-Based Systems, LNCS*, vol. 3137, pp. 235–244, 2004. xx, 25

[30] W. Schwinger, C. Grn, B. Prll, W. Retschitzegger, and A. Schauerhuber, "Context-awareness in mobile tourism guides - a comprehensive survey," *Technical Report*, July 2005. xx, 25

[31] M. Dunlop, B. Elsey, and M. Masters, "Dynamic visualisation of ski data: a context aware mobile piste map," in *In Proceedings of the 9th international conference on Human computer interaction with mobile devices and services (MobileHCI '07)*, (New York, USA), 2007. xxi, 25

[32] D. Black, N. J. Clemmensen, and M. B. Skov, "Supporting the supermarket shopping experience through a context-aware shopping trolley," in *In Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7 (OZCHI '09)*, (New York, NY, (USA)), 2009. xxi, 25

[33] A. Rakotonirainy and N. Lehman, "Augmenting a museum visitor's tour with a context aware framework," in *in Proceedings of the 1st International Workshop on Ubiquitous Computing*, (Porto, Portugal), 13-14 April 2004. xxi, 26

[34] M. I. of Context Awareness in the Design of 3GPP Conversational Services, "F. toutain and a. bouabdallah and r. zemek and c. daloz," in *In proceeding of the 6th International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST'12)*, (Paris, France), 2012. 26

[35] G. T. 24.604, "Communication diversion (cdiv) using ip multimedia (im) core network (cn) subsystem; protocol specification - v.11.2.0," 2012. 26

[36] B. Chihani, E. Bertin, and N. Crespi, "A comprehensive framework for context-aware communication services," in *In Proceedings of the 15th International Conference on Intelligence in Next Generation Networks (ICIN)*, (Berlin, Germany), October 2011. 27

[37] A. Beach, M. Gartrell, X. Xing, R. Han, Q. Lv, S. Mishra, and K. Seada, "Fusing mobile, sensor, and social data to fully enable context-aware computing," in *HOTMOBILE*, (Annapolis, Maryland (USA)), 2010. 28

[38] K. Hamadache, E. Bertin, A. Bouchacourt, and I. Benyahia, "Context-aware communication services: an ontology based approach," in *2nd International Conference on Digital Information Management (ICDIM'07)*, (Lyon, France), October 2007. 28, 60

[39] T. Yamabe, A. Takagi, and T. Nakajima, "Citron: A context information acquisition framework for personal devices," in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, (Hong Kong, China), 2005. 28

[40] B. Saghir and N. Crespi, "A generic layer model for context-aware communication adaptation," in *In: IEEE Wireless Communications and Networking Conference, WCNC 2008, vol. 9(1)*, p. 30273032, 2008. 28

[41] S. Herborn, H. Petander, and M. Ott, "Predictive context aware mobility handling," in *International Conference on Telecommunications*, June 2008. 29

[42] N. Eagle and A. S. Pentland, "Reality mining: sensing complex social systems," *Personal Ubiquitous Computing*, vol. 10, pp. 255–268, March 2006. 29

[43] E. I. Tatli, *Security in context-aware mobile business applications.* PhD thesis, University of Mannheim, 2009. 29

[44] K. Schreiner, "Where we at? mobile phones bring gps to the masses," *IEEE Computer Graphics and Applications*, vol. 27, May-June 2007. 29

[45] W. B. Powell and T. A. Carvalho, "Dynamic control of logistics queueing networks for large-scale fleet management," *Transportation Science*, vol. 32, May 1998. 29

[46] T. Hardie, A. Newton, H. Schulzrinne, and H. Tschofenig, "Lost: A location-to-service translation protocol," *RFC 5222*, August 2008. 29

[47] J. Choi and J. Moon, "Myguide: A mobile context-aware exhibit guide system," in *In Proceedings of the International Conference on Computational Science and Its Applications (ICCSA'08)*, (Perugia, Italy), June 30 July 3 2008. 30

[48] B. Fleischmann, S. Gnutzmann, and E. Sandvoss, "Dynamic vehicle routing based on online traffic information," *Transportation Science*, vol. 38, November 2004. 30

[49] A. Bahrami, J. Yuan, R. S. Paul, and N. Shadbolt, "Context aware information retrieval for enhanced situation awareness," in *Proceedings of IEEE Military Communications Conference (MILCOM'07)*, (Orlando, Florida), pp. 29–31, October 2007. 30

[50] B. Chihani, E. Bertin, and N. Crespi, "A graph-based context modelling approach," in *In proceedings of the 4th International Conference on SmArt COmmunications in NEtwork Technologies (SaCoNeT'13)*, (Paris, France), June 17-19 2013. 30, 129

[51] M. D. Choudhury, H. Sundaram, A. John, and D. Seligmann, "Context aware routing of enterprise user communications," in *Fifth annual IEEE international Conference on Pervasive Computing and Communications Workshops (PerCom'07)*, (White Plains, NY, USA), 19-23 March 2007. 30

[52] F. Alt, A. S. Shirazi, M. Pfeiffer, P. Holleis, and A. Schmidt, "Taximedia: An interactive context-aware entertainment and advertising system," *Lecture Notes in Informatics, GI Jahrestagung*, vol. 154, 2008. 30

[53] H. Liu and M. Rauterberg, "Context-aware in-flight entertainment system," in *In proceedings of the 12th International Conference on Human-Computer Interaction (HCI'07)*, (Beijing, China), 22-27 July 2007. 30

[54] M. Fayad and D. C. Schmidt, "Object-oriented application frameworks," *Communications of the ACM Magazine*, vol. 40, October 1997. 31

[55] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. 31, 94

[56] C. Harr, T. Finin, and A. Joshi, "An intelligent broker for context-aware systems," in *Adjunct Proceedings of Ubicomp 2003*, (Seattle, Washington, USA), October 12-15 2003. 31

[57] T. Gu, H. K. Pung, and D. Q. Zhang, "A middleware for building context-aware mobile services," in *IEEE 59th Vehicular Technology Conference (VTC04)*, (Milan, Italy), 2004. 31, 55, 61

[58] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007. 31

[59] S. Loke, *Context-Aware Pervasive Systems: Architectures for New Breed of Applications*. Auerbach Publications, 2006. 31

[60] L. B. Wilson and R. G. Clark, *Comparative Programming Languages*. Wokingham, England, Addison-Wesley, 1988. 35

[61] D. Morley and C. S. Parker, *Understanding Computers 2009: Today and Tomorrow*. Cengage Learning, 2009. 35

[62] Android, "Web page," March 2013. http://www.android.com/. 35

[63] V. Lindberg, *Intellectual Property and Open Source: A Practical Guide to Protecting Code*. O'Reilly Media, Inc., 2009. 35

[64] Nuanc, "Dragon mobile sdk," March 2013. http://www.nuance.com/for-developers/dragon-mobile-sdk/index.htm. 35

[65] E. Visser, "Webdsl: A case study in domain-specific language engineering," *Generative and Transformational Techniques in Software Engineering, Springer Berlin Heidelberg*, pp. 291–373, 2008. 35

[66] Rails, "Ruby on rails," March 2013. http://rubyonrails.org/. 36

[67] C. Britton and P. Bye, *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*. Pearson Education, 2004. 36

[68] K. Gajos, H. Fox, and H. Shrobe, "End user empowerment in human centered pervasive computing," in *Proceedings of Pervasive 2002*, (Cambridge, MA, USA: MIT AI Lab), 2002. 38

[69] F. P. Brooks, "No silver bullet: Essence and accidents of software engineering," *IEEE Computer*, vol. 20, pp. 10–19, April 1987. 38

[70] G. W. Musumba and H. O. Nyongesa, "Context awareness in mobile computing: A review," *International Journal of Machine Learning and Applications*, vol. 2, no. 1, 2013. 42, 49

[71] Y. Ho, Y. Wu, and M. Chen, "Plash: a platform for location aware services with human computation," *IEEE Communications Magazine*, vol. 48, no. 12, pp. 44–51, 2010. xxii, 50

[72] D. R. de Almeida, C. S. Baptista, E. da Silva, C. E. C. Campelo, H. F. de Figueirdo, and Y. A. Lacerda, "A context-aware system based on service-oriented architecture," in *Proceedings of 20th International Conference on Advanced Information Networking and Applications (AINA'06)*, (Vienna, Austria), September 2006. 53

[73] H.-L. Truong, L. Juszczyk, S. Bashir, A. Manzoor, and S. Dustdar, "Vimoware - a toolkit for mobile web services and collaborative computing," in *Special session on Software Architecture for Pervasive Systems, 34th EUROMICRO Conference on Software Engineering and Advanced Applications*, (Parma, Italy), 3 - 5 September 2008. xxii, 59

[74] P. Gutheim, "An ontology-based context inference service for mobile applications in next-generation networks," *IEEE Communications Magazine*, vol. 50, no. 1, pp. 6 –66, 2011. 60

[75] Y. Oh, J. Han, and W. Woo, "A context management architecture for large-scale smart environments," *IEEE Communications Magazine*, vol. 48, no. 3, pp. 118–126, 2010. 61

[76] S. McKeever, J. Ye, L. Coyle, and S. Dobson, "A context quality model to support transparent reasoning with uncertain context," in *In proceedings of the 1st International Workshop on Quality of Context (QuaCon)*, (Stuttgart, Germany), 2009. 62

[77] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. 90

[78] D. D. Corkill, "Collaborating software: Blackboard and multi-agent systems & the future," in *In Proceedings of the International Lisp Conference*, (New York, USA), October 2003. 90

[79] T. Weilkiens, *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. Morgan Kaufmann, 2011. 91

[80] N. . SEMATECH, *e-Handbook of Statistical Methods*. 2012. http://www.itl.nist.gov/div898/handbook/. 93

[81] V. G. Jilles and J. Bosch, "On the implementation of finite state machines," in *Variability in Software Systems the Key to Software Reuse*, 2000. 93

[82] R. Copeland, *Converging NGN Wireline and Mobile 3G Networks with IMS*. CRC Press, 2008. 94

[83] R. Stevens, *UNIX Network Programming: Inter-process Communications*, vol. 2. Prentice Hall, 1999. 99

[84] M. Bell, *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. New York: John Wiley & Sons, 2008. 99

[85] R. Meier, *Professional Android 4 Application Development*. John Wiley & Sons, April 5 2012. 102

[86] D. Gourley, B. Totty, M. Sayer, A. Aggarwal, and S. Reddy, *HTTP: The Definitive Guide*. O'Reilly Media, Inc., October 2009. 102

[87] J. Hudson, J. Christensen, W. Kellogg, and T. Erickson, "I'd be overwhelmed, but it's just one more thing to do: availability and interruption in research management," in *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves (CHI'02)*, (Minneapolis, Minnesota, USA), 2002. 118

[88] B. Chihani, E. Bertin, F. Jeanne, and N. Crespi, "Hep: Context-aware communication system," *International Journal of New Computer Architectures and their Applications (IJNCAA'11)*, vol. 1, May 2011. 118

[89] M. Gortz, R. Ackermann, J. Schmitt, and R. Steinmetz, "Context-aware communication services a framework for building enhanced ip telephony services," in *In Proceedings of the 13th International Conference on Computer Communications and Networks (ICCCN'04)*, (Chicago, IL), October 2004. 120

Glossary

AI Artificial intelligence. 20

API Application Programming Interface. 31, 32, 50–52, 56, 73, 112, 123, 140, 151

BPM Business Process Management. 22

BSCS Business Social Communication Services. 4

CAAB Context-Aware Address Book. 79, 80, 112

CMS Context Management Systems. 31, 32, 67

CPDL Context Processing Definition Language. 81, 113

CRM Customer Relationship Management. 82

CxB Context Broker. 64, 66, 76

CxC Context Consumer. 64, 66, 90, 102, 103, 110

CxP Context Provider. 64–67, 74–76, 90, 102, 110

DTD Document Type Definition. 76, 100, 101, 103

ESD Enterprise Social Graph. 129, 133, 134

GEF Graphical Editing Framework. 136

GHP General Human Profile. 16

GPS Global Positioning System. 17, 19, 25, 28, 29, 31–33, 35, 44, 70, 75, 88, 89, 92, 115

GUI Graphical User Interface. 115

HCI Human-Computer Interaction. 115

HEP enHancEd un-interruPtibility. 118, 121–123, 127

IM Instant messaging. 60, 118, 122–127

IMS IP Multimedia Subsystem. 81
IT Information Technology. 86, 128

JDBC Java Database Connectivity. 51
JDK Java Development Kit. 87
JMS Java Message Service. 51

LBS Location-based service. 50, 53, 54
LDAP Lightweight Directory Access Protocol. 129, 130
LDSQ Location-dependent Spatial Query. 50, 51
LED light-emitting diode. 24

M2M Machine to Machine. 68, 115
MDA Model Driven Architecture. 21

OWL Web Ontology Language. 21, 53, 55–57

PBX private branch exchange. 119
PDP Policy Decision Point. 77
PEP Policy Enforcement Point. 77
POI Point of interest. 53, 55

RDF Resource Description Framework. 21, 57, 71
REST Representational State Transfer. 50, 51, 69, 90

SDK Software Development Kit. 85–89, 91–100, 102, 108, 109
SIP Session Initiation Protocol. 72, 113
SMA Simple Moving Average. 74
SOAP Simple Object Access Protocol. 59
SQWRL Semantic Query-Enhanced Web Rule Language. 21
SWRL Semantic Web Rule Language. 21

UCEM Unified Communications and Emerging Markets. 3
UCMA Unified Communications Managed API. 112, 123
UML Unified Modeling Language. 22, 87, 93, 124, 125, 131
URI Uniform Resource Identifier. 113

V2I vehicule-to-infrastructure. 50

V2V vehicule-to-vehicule. 50

WEFEE Work Environment for employees empowerment. 4

XML eXtensible Markup Language. 22, 50, 59, 63, 83, 87–89, 92, 94, 95, 98, 100, 102, 103, 105–107, 111, 130, 131, 135, 136

XMPP eXtensible Messaging and Presence Protocol. 22