CONTEXTUALIZED AND PERSONALIZED LOCATION-BASED SERVICES

THÈSE N^{O} 3896 (2008)

PRÉSENTÉE LE 22 FÉVRIER 2008

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Laboratoire de bases de données

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Shijun YU

BSc in Computer Science and Technology, Northern Jiaotong University, Chine
et de nationalité chinoise

acceptée sur proposition du jury:

Prof. B. Faltings, président du jury
Prof. S. Spaccapietra, directeur de thèse
Prof. K. Aberer, rapporteur
Prof. N. Cullot, rapporteur
Prof. E. Zimányi, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL
2008

To my mother Wenhui Wang, my father Yongnian Yu,

and my sister Shikun Yu.

Acknowledgements

I would like to thank many people during my PhD journey, who accompanied with me for exciting and joyful, hard and sad time.

My first gratitude goes to my mentor Prof. Stefano Spaccapietra, who gave me the great opportunity to work on the interesting research topic and in LBD, the amiable working atmosphere. His guidance, care, humor, and optimistic attitude to the life and work intensively impress me and encourage me.

I also thank Prof. Boi Faltings to president over the doctoral defense, and Prof. Karl Aberer, Prof. Nadine Cullot and Prof. Esteban Zimányi for their kind participation to my defense and their interests on my work.

I would like to thank all my friends and colleagues in LBD, Christelle, Fabio, Tonho, Olexandr, Hassina, Anastasiya, Othman, Marlyse, Chiara, Zhixian, and Jena for their help and support. I gratefully appreciate Prof. Christine Parent for her encouragement and advice for my work, and her concern on my life. I also would like to present my gratitude to Prof. Nadine Cullot, Dr. Christelle Vangenot and Dr. Fabio Porto for their constructive discussion and advice in my work. In addition, I would appreciate my remote mentors, Dr. Allan Cheng and Dr. Micki Wiersma for their valuable experiences sharing and advices on my PhD path.

I greatly appreciate for my friends in Switzerland and in the remote, particularly, Beilu, Bingjiang, Dongbin, Dongyu, Gloria, Hai, Hong, Huan, Hongze, Leen, Li, Lina, Ling, Qian, Qing, Quan, Ruohua, Shili, Wanna, Wanjun, Xiaolei, Xin, Yan, Yanjie, Yang, Ye, Yi, Yongcheng, Yonggeng, You and many other friends. I will never forget the cheerful parties in summer, warm Christmas dinners, our footprints in Valais valleys, our explorations in Scandinavia, and pleasant French-Chinese tandem nights...

The last but not least, I would gratefully thank to my family, my parents and my sister, for their endless trust, support and love, as well as critiques and advices whenever I was lost or helpless in my life. This doctoral dissertation is dedicated to them.

Abstract

Advances in the technologies of smart mobile devices and tiny sensors together with the increase in the number of web resources open up a plethora of new mobile information services where people can acquire and disseminate information at any place and any time. *Location-based services* (LBS) are characterized by providing users with useful and local information, i.e. information that belongs to a particular domain of interest to the user and can be of use while the user remains in a particular area. In addition, LBS need to take into account the interactions and dependencies between services, user and context for the information filtering and delivery in order to fulfil the needs and constraints of mobile users. We argue that consequently it brings up a series of technical challenges in terms of data semantics and infrastructure, context-awareness and personalization, as well as query formulation and answering etc. They can not be simply extended from existing traditional data management strategies. Instead, they need a new solution.

Firstly, we propose a semantic LBS infrastructure on the basis of the modularized ontologies approach. We elaborate a core ontology which is mainly composed of three modules describing the services, users and contexts. The core ontology aims at presenting an abstract view (a model) of all information in LBS. In contrast, data describing the instances (of services user and actual contextual data) are stored in three independent data stores, called the service profiles, user profiles and context profiles. These data are semantically aligned with the concepts in the core ontology through a set of mappings. This approach enables the distributed data sources to be maintained in a autonomous manner, which is well adapted to the high dynamics and mobility of the data sources.

Secondly, we separately address the function, features, and our modelling approach of the three major players, i.e. service, context and user in LBS. Then, we define a set of constructs to represent their interactions and inter-dependencies and illustrate how these semantic constructs can contribute to personalized and contextualized query processing. Service classes are organized in a taxonomy, which distinguishes the services by their business functions. This concept hierarchy helps to analyze and reformulate the users' queries. We introduce three new kinds of relationships in the service module to enhance the semantics of interactions and dependencies between services. We identify five key components of contexts in LBS and regard them as a semantic contextual basis for LBS. Component contexts are related together by specific composition relationships that can describe spatio-temporal constraints. A user profile contains personal information about a given user and possibly a set of self-defined rules, which offer hints on what the user likes or dislikes, and what could attract him or her. In the core ontology clustering users with common features can help the cooperative query answering. Each of the three modules of the core ontology is an ontology in itself. They are inter-related by relationships that link concepts belonging to two different modules. The LBS fully benefits from the modularized structure of the core ontology. It allows restricting the search space, as well as facilitating the maintenance of each module.

Finally, we studied the query reformulation and processing issues in LBS. How to make the query interface tangible and provide rapid and relevant answers are typical concerns in all information services. Our <what, when, where, what-else> query format not only fully obeys the 'simple, tangible and effective' golden-rules of user-interface design, but also satisfies the needs of domain-independent interface and emphasizes the importance of spatio-temporal constraints in LBS. With pre-defined spatio-temporal operators, users can easily specify in their queries the spatio-temporal availability they need for the services they are looking for. This allows eliminating most of irrelevant answers that are usually generated by keyword-based approaches. Constraints in the various dimensions (what, when, where and what-else) can be expressed by a conjunctive query, and then be smoothly translated to RDF-patterns. We illustrate our query answering strategy by using the SPARQL syntax, and explain how the relaxation can be done with rules specified in the query relaxation profile.

Keywords: Location-based services, ontology, context-awareness, personalization.

Résumé

L'avancée des technologies mobiles et des capteurs miniatures conjointement avec l'augmentation du nombre de ressources disponibles sur Internet a favorisé le développement de nouveaux services géolocalisés qui permettent l'acquisition et la dissémination d'informations à tout moment et à tout endroit. Les *services géolocalisés* sont caractérisés par leur capacité à fournir à leurs utilisateurs des informations utiles et localisées, c'est-à-dire des informations correspondant au domaine d'intérêt de l'utilisateur et qui lui sont utiles par rapport à sa localisation géographique actuelle. De plus, les services géolocalisés doivent prendre en compte les interactions et les dépendances entre les services, l'utilisateur et son contexte afin de filtrer et de retourner de l'information satisfaisant les besoins et contraintes des utilisateurs mobiles. De ce fait, cela induit différents défis techniques concernant la sémantique et l'architecture des données, la personnalisation et la prise en compte du contexte, ainsi que la définition et l'exécution des requêtes. En effet, le développement de services géolocalisés nécessite de définir de nouvelles solutions allant au-delà de l'extension simple des stratégies traditionnelles de gestion et d'interrogation de données.

Nous proposons tout d'abord une architecture de services géolocalisés reposant sur une ontologie modulaire. Nous avons élaboré une ontologie centrale composée principalement de trois modules décrivant respectivement les services, les utilisateurs et les contextes. L'ontologie centrale a pour objectif de présenter une vue abstraite de toutes les informations décrites dans l'infrastructure de services géolocalisés. De leur côté, les données décrivant les instances de services, du contexte et des utilisateurs sont stockées dans trois banques de données indépendantes, appelées profils de services, profils d'utilisateurs et profils de contextes. Ces données sont mises en correspondance avec les concepts de l'ontologie centrale via un ensemble de règles de correspondance. Cette approche, qui permet de maintenir chaque source de données de façon autonome, est parfaitement adaptée à la mobilité et à la dynamique des sources de données des services géolocalisés.

Ensuite, nous avons étudié séparément les fonctionnalités, les caractéristiques et la modélisation des trois principaux acteurs de notre approche, c'est-à-dire les services, les utilisateurs et le contexte. Nous avons défini un ensemble de concepts pour représenter leurs interactions et interdépendances, puis nous avons montré en quoi ces concepts sémantiques contribuent à linterrogation contextuelle et personnalisée des données. Les classes de service sont décrites dans une taxonomie qui distingue les services sur la base de leurs fonctionnalits commerciales. Cette hiérarchie de concepts facilite l'analyse et la reformulation des requêtes utilisateurs. Nous avons introduit trois nouveaux types de relations dans le module des services afin de mettre en évidence la sémantique des interactions et des dépendances entre services. Nous avons identifié cinq composants clés définissant la notion de contexte; ils constituent la base de notre approche contextuelle pour les services géolocalisés. Ces contextes peuvent être reliés entre eux par des relations spécifiques de composition qui permettent de décrire des contraintes spatio-temporelles. Le profil utilisateur renferme les informations personnelles relatives à un utilisateur donné et éventuellement un ensemble de règles (définies par l'utilisateur lui-même) décrivant ce qu'il aime ou n'aime pas et ce qu'il pourrait désirer. Dans l'ontologie centrale, les utilisateurs sont regroupés selon leurs caractéristiques communes, ce qui facilite le traitement coopératif des requêtes. Chacun des trois modules de l'ontologie centrale est lui-même une ontologie. Ils sont reliés par des relations qui mettent en correspondance des concepts appartenant à deux modules différents. Notre approche de services géolocalisés est grandement facilitée par la structure modulaire de notre ontologie centrale. Elle permet en effet de restreindre l'espace de recherche lors des requêtes et aussi de faciliter la maintenance de chaque module.

Enfin, nous avons étudié la reformulation et l'exécution des requêtes pour les services géolocalisés. Dans une telle approche, les défis principaux concernent lergonomie de l'interface de requêtes ainsi que le temps de réponse et la pertinence des données retournées. Notre format de requête, ≪quoi, quand, où, quoi-dautre≫, répond non seulement aux trois règles d'or de la conception d'interface utilisateurs, "simplicité, tangibilité et efficacité", mais il satisfait aussi les besoins d'indépendance de l'interface par rapport au domaine et il met en valeur l'importance des contraintes spatio-temporelles pour les services géolocalisés. Lors de la formulation des requêtes, les utilisateurs peuvent ainsi facilement spécifier la validité spatio-temporelle désirée pour les services recherchés à l'aide d'opérateurs spatio-temporels. Cela permet d'éliminer les réponses non pertinentes qui sont généralement générées par les approches basées sur la spécification de mots clés. Les contraintes définies dans les différentes dimensions (quoi, quand, où, quoi-dautre) peuvent être exprimées à l'aide d'une requête formée de conjonctions et ainsi être traduite facilement en patrons RDF. Nous avons illustré notre stratégie de traitement de requêtes à l'aide de la syntaxe SPARQL. Enfin nous avons expliqué comment la relaxation des requêtes peut être réalisée avec des règles spécifiées dans un profil de relaxation de requêtes.

Mots-clés: services géolocalisés, ontologie, personnalisation, prise en compte du contexte.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

1.1 IT Support for Mobility

The quest for mobility has always been a characteristic of humans. Our ability to move has gradually evolved from using our legs to using animals, animal-driven devices (e.g. chariots), self-driven devices (this is exactly the meaning of "automobile"), mass transportation systems (e.g. trains and boats), and finally ending up with the ability to flow through atmosphere (planes and space rockets). But while mobility increased, complexity of the world also increased and our knowledge about the world has become by far inadequate to support our activities on the move. In particular, we recently became so information and communication addicted that it is almost beyond imagination to think that being on the move would mean being deprived of the ability to communicate and to get access to information available from the information technology (IT) world. Fortunately, IT development has been able to start matching our requirements. Mobile and wireless data networks (e.g. GPRS[1] and UTMS[2]) [Tis01] provide the communication infrastructure that allow staying connected if not everywhere at least in many places around the world. Global positioning systems (GPS), mobile phones and personal digital assistants (PDAs) provide the devices that users keep with them and cherish to be able to jump into the IT world whenever they want. Sensor networks [ZG04] are the latest addition to this panoply of innovative technologies and provide up-to-date information that may be quite useful in adjusting information to current situations (e.g., to suggest the best itinerary to reach the airport given current traffic conditions). The convergence of so many technological advances is revolutionizing human's socio-economical life. The computing environment has added to the traditional desktop-bound fashion a new ubiquitous computing [Wei93] mode of operation. At first, ubiquitous computing just took advantage of wireless services to make available to moving users the traditional facilities of desktop computing, such as access to web pages and emails. It has since become clear that ubiquity leads in fact to a larger spectrum of facilities [GF05] up to services offering intelligent and personalized services to people on the move, wherever they are and whenever they ask for service. Faster than expected, many science-fiction scenarios are turning into reality, e.g. intelligent meeting room, portable hotel reservation assistance,

[1] GPRS: acronym for General Packet Radio Service.
[2] UMTS: acronym for Universal Mobile Telecommunications System.

continuous heath-care monitoring, environmental monitoring, etc. Among the novel services, Location-Based Services(LBS) aim at providing moving users with "right on the spot" information, i.e., information that relates to the area where the user is currently located, belongs to a particular domain of interest to the user and can be of use while the user remains in the area. LBS are becoming increasingly popular as they correspond to a real need that everyone of us can experience. Moreover, they are already available in some places, which facilitates their dissemination through public media to the general public. LBS provide the framework for this dissertation.

This chapter briefly introduces different categories of services using locational information. The spread of positioning technology paved the path to nowadays and future LBS. We then sketch how LBS were defined by previous researchers and give our definition. We highlight what distinguishes LBS from other mobile services paradigms, and the main challenges in designing and implementing LBS. Finally, the contributions and the organization of the dissertation are given.

1.2 Positioning Services

Location-based Services have not been invented from scratch. They stem from earlier *positioning services*[3] which were developed for military use in the 1970's in the USA. Positioning services respond to the most basic requirement related to mobility: the need to know where moving people and objects are currently located. Their aim and scope is answering requests such as "Where am I?" or "Where is XXX?". This functionality has become available since the earliest GPS (Global Positioning Services) systems. Through the measurement and computation of satellites signals received by the GPS device, GPS-enabled positioning services are nowadays able to offer good performance in most outdoor services. This enables numerous interesting applications, e.g. computer-aided navigation [Gar04], road emergency assistance, and tracking the target assets in fleet management. However, positioning precision and timeliness largely degrade when deployed in indoors or urban area due to the well-known effect 'urban canyon'. In such cases, other positioning technologies, such as GSM, Bluetooth, and RFID, are utilized to provide complementary positioning services. One typical example is the infrared-based Active Badge system [WHFG92].

A positioning service provides the location of an object according to a given format and a precision level (resolution). It can also provide more sophisticate location utility services. An overall review of the characteristics, precision and applicable coverage scope of fundamental positioning technologies, such as GPS, DGPS, GSM, E-OTD can be found in [DB03].

In an effort to prevent or solve heterogeneity issues, the OpenLS[4] initiative promotes standards to facilitate and consolidate the interoperation of geo-spatial data and geo-processing of location services, especially for geo-coding and reverse geocoding, route determination, and map/features display etc. By definition any LBS, whether its infrastructure is simple or complex, or whether it supports indoor or outdoors or both [ZGL03],

[3]The term 'location services' is sometimes used as a synonym for 'positioning services'. In our work we prefer using 'positioning services' to avoid any confusion between 'location service' and 'location-based services'.

[4]OpenLS (Open Location Services) is a proposal for location-related standards initiated by OGC (Open Geospatial Consortium). For more details, see http://www.opengeospatial.org/functional/?page=ols.

includes at least one 'positioning service' embedded in or associated with it. Standard functionalities therefore appeal to LBS for enhancing the spatial-awareness, and the interoperability of heterogeneous spatial data involving end-users and data services in LBS.

During the last decade a series of novel technologies (improved networking capabilities and interface characteristics [SW03]) promoted the use and adoption of mobile handheld device, further popularizing positioning services. The prevalence of positioning services makes it commercially feasible to obtain and manipulate the location of mobile objects so as to boost the development of more sophisticate mobile services. In order to better understand the state of art of current 'information services' in mobile environment, we discuss some of these paradigms in terms of their basic states and functionalities in next section.

1.3 Information Dissemination Services

Most services are geared towards information dissemination, using either one of two fundamental dissemination paradigms: on demand access (also called pull mode), and broadcast (also called push mode). Obviously, hybrid approaches are also available, providing information according to either one of the two modes depending on information content and user selections.

In the *on-demand access* mode, the user has the active role to initiate the interaction with the service by submitting a request to the server. The server processes the queries and sends the answer to the requester. But its performance could rapidly decline due to the sudden increase of workload, e.g. immense requests about the latest ranking of *World Cycling Championship*. This mode suits the need for ad hoc queries, assuming the answer to the query can be computed in reasonable time (e.g. a few seconds). Its drawback in current mobile systems is the same as in desktop systems: information flooding and poor relevance of the retrieved information whenever the request is not looking for a precise piece of information. For example, a query to retrieve the departure times for trains from Lausanne to Geneva in between 2pm and 3pm on a weekday will return precisely what is being requested. Instead, a Google-like query to retrieve information about St John's cathedral will return a list of all web pages that have to do with cathedrals, John, and street or saints (St).

The *information broadcast* mode offers an alternative way to efficiently disseminating information to multiple users at a fixed cost. In this mode the user is passive. Information automatically comes to her/him without having to be solicited. More precisely, *information broadcasting* denotes the communication technique where emitting sources disseminate information to the surrounding world, without targeting a specific individual receiver. Receivers are active mobile devices located within the range of emission and which have the capability to capture the information. The typical example is the welcome message delivered by mobile service providers when a cellphone user roams to another country. The message delivery occurs without the user making any request, i.e. in a full push-mode.

In contrast to the current *4A* paradigm (i.e. Anybody, Anytime, Anywhere, Anything) that has enthusiastically supported the spread of ubiquitous computing and characterizes services such as browsing and searching on the web, the information broadcasting paradigm can be characterized by the combination '*Anybody, Anytime, Anywhere, Something*'. *Anywhere* here only indicates any point within the scope of signal emission, mainly relying on the type of wireless networking (e.g. wireless cellular networks, wireless LAN, wireless ATM and Wireless PAN/BAN). *Something* denotes a content that is defined and delivered uniquely by the service provider. Most of the benefit is for the disseminator rather than for the receiver. For example, broadcasting of commercial messages to cellphones within a shopping mall can be very effective to attract more customers to a shop. In a shopping mall a large amount of information sources may be available with a very limited emission range. But from the user viewpoint shopping in the mall may become annoying to the point that they switch off their cellphone (yet modern broadcasting to cellphones has enormous advantages versus traditional broadcasting via loudspeakers). The unfortunate best and worst example is spam on email. Broadcasting offers the advantage of the simplicity of its operation but is no remedy to insufficiency and irrelevancy of information reaching the user.

A compromise between simplicity of broadcasting and user interest in the broadcasted information is realized through the publish/subscribe strategy [HGM04]. Information services using this strategy improve the information relevancy to recipient's needs by matching new information against user's predefined interests. In this paradigm, the receivers still passively wait and receive information published by providers. Thus, its design is information-driven and service-oriented rather than user-motivated. GUIDE [DCMF99] push-style information dissemination is an example of this strategy. However, information dissemination in publish/subscribe broadcasting aims at delivering some specific information (e.g. the advertisement message from certain shops) to some specific users (those who have subscribed to receive this type of information). This strategy is currently very popular for information dissemination via email (users subscribe to a distribution list) and via the web (chats, blogs and alike). It is not specifically related to moving users and mobile information systems. It can be characterized as '*Somebody, Anytime, Anywhere, Something*'. A mobile variant of publish/subscribe can be illustrated using the same shopping mall example. Imagine that the various shops broadcast (within a limited range to avoid interferences) their advertisement messages to people in their vicinity. Here the action of subscribing is replaced by the action to move into the emission range of a shop. This is an '*Anybody, Anytime, Somewhere, Something*' mode of operation. Imagine now that users' cellphones are equipped with a system that allows filtering incoming messages based on users' predefined interests. Whether the filtering is done by the shop or by the user, and assuming messages are only broadcasted by a shop during its opening hours, the operation mode becomes 4S: '*Somebody, Sometime, Somewhere, Something*'. This can be an interesting complement to the current 4A paradigm and 'push' paradigms. Unfortunately, it is getting too little attention from both industry and academia.

An interesting alternative to the publish/subscribe paradigm for a '*Somebody, Anytime, Anywhere, Something*' mode of operation is the publish/publish mode. The information provider still has the active role to "publish" new information. On the other side, users play the same active role by "publishing" their profile, i.e. the definition of the type of information they are interested in. A broker matches information characteristics (data profile) with user profiles and whenever a match reaches a given threshold the information is transferred to the user in a push mode. The broker can be an autonomous software or part of the provider system. This mode, where both users and providers become publishers, enables active and asynchronous matching between the current users' specifications (understood as potential automatic subscriptions to whatever matches the specification) and existing information made publicly available by any information provider. It is a first step towards more advanced techniques aiming at personalization and intelligence of information services. Though such broadcasting paradigms enable the system to deliver the timely information to relevant users, they are not tailored to answer ad-hoc requests when the user is on the move. Such requests call by definition for a user-pull mode.

The aim to achieve more personalized information services is taken further by context-aware computing solutions [CDM$^+$00]. These embody paradigms where generic contextual information, not merely location or user profile, can be specified and used to increase relevance in providing information or services [DA99]. The definitions and categorizations of context can be found in e.g. [SAW94], [DA99], [vBFA05]. They are usually coupled with sensor networks, which collect, process, integrate and supply basic contextual data that is mainly not user-related. From the information dissemination perspective, context-aware systems enable to use and/or adapt user's current context to present information (e.g. show a list of printers close to the user walking in the building), or to trigger a service execution (e.g. to send the printing job to the nearest printer for the user). However, the information delivered in most context-aware systems is accordingly narrow and application-specific.

1.4 Mobile Information Services and LBS

We refer to 'mobile information services' in general as information services able to provide information support to roaming users through their mobile devices. Mobile information services have much in common with e-services [RK03] in terms of their network-enabled modality and mobility. But, unlike most e-services, they are not always transactional. Mobile information services can be implemented using any of the information dissemination paradigms we have presented in the previous section. We focus hereinafter on the push mode, i.e. on responding to user's ad-hoc requests for information. Moreover, we focus on dealing with requests from mobile users in search of information about the geographical area in which they happen to be. This is typically the case for users on vacation (tourists) or on a business trip (mobile employees). In short, we focus on Location-based services. The difference between generic mobile information services and LBS is that for the latter knowledge of current time and user's current location and its evolution is a critical element ruling how the LBS will perform the information extraction process in response to user's requests. The former do not necessarily use the time and location information. For example, providing web access via a cellphone qualifies as a mobile information service but not as a location-based service. Notice that the user location is an input parameter to the interaction with the LBS. It does not need to be the location where the user physically is at the moment. This entails that LBS can be tuned as web services, i.e. they can provide the same service to users accessing from anywhere on the web, as long as the accessing user defines the location where (s)he virtually wants to be. For example, while in New York a user can ask the query "assuming I am in Lausanne, place St. François, at 4pm on a weekday, is there a close-by museum I still would have enough time to visit on the same day?". The answer could be "Museums close at 5pm, so you may have some time to visit the Design Museum or Historical Museum, both located near the cathedral,. Alternatively, as you are interested in art, you may visit the art galleries that are within 5 minutes walk from St François. On Friday you may instead visit the Musée de l'Elisée (photography) who remains open till 8pm on Fridays".

Location-based Services, also denoted as *location-dependent services* and *location-aware services* have become an increasingly popular topic, and both academia and industry are intensively investigating their development and applications.

Location-based services hold the potential to revolutionize many fields of our socio-economical life, from environmental monitoring and conservation, to manufacturing and business assets management, logistics management, to automation in the home appliances and health-care. Its development can also be extended to entertainment markets, such as online games and virtual museum visit. Surveys on application of LBS can be found in the literature [Spi04].

Most researchers have a common but loose definition about LBS, i.e. *a service provided to mobile users based on their geographical location.* However, such a definition is vague and insufficient to characterize LBS. In addition, although location and time often are primary determinants in the service provided by LBS, we believe personalization and context-awareness are also very significant in LBS. Although these two aspects are very generic and may apply to any kind of information service, we feel in the short-term future they will represent a mandatory feature of successful LBS. Our rationale relies on the fact that interactions between mobile users and LBS are characterized by the need to reach a satisfactory response to user requests in a very short time. A mobile user is not likely to go through much iteration to find out relevant information from the flood of information that is returned in the absence of personalization and context-awareness. As a counter-example, it is known that going to the web to find some information, while appearing at first as a matter of a few minutes if not seconds, turns out in fact to easily take up to half an hour and a lot of page browsing before a satisfactory answer is determined. This iterative browsing is definitely a kind of service that is irrelevant to mobile users. Hence, realizing what LBS are and what functionalities LBS would provide are only the first step towards designing a practical and intelligent LBS infrastructure. In this section, we will investigate notations about LBS, explore the functionalities of existing LBS, and further discuss its formality and specificity that make LBS different from other mobile services.

1.4.1 Location-based Services Evolution

Active Badge [WHFG92], developed at Olivetti Research Lab in 1992, is generally acknowledged as the first research experimentation of LBS. It took advantage of the availability of new positioning services to improve the efficiency of an in-house call forwarding application. Knowledge of the current locations of mobile employees in the building made it possible to forward calls directly to the targeted employee rather than generically broadcast the call through the entire building to make sure it reached the employee. Similarly, in project GUIDE and CyberGuide, proximity-based tourism information can be delivered to the mobile users. Other industrial products include friend-finder at AT&T, WebSphere-based LBS at IBM and MapPoint-enabled LBS at Microsoft. In practice, at NTT-DoCoMo, i-area has been able to operationally deliver users a broad spectrum of facilities nearby according to their current location. These designs exemplified the same denotation as [SV04], 'LBS are applications integrating geographic location (i.e. spatial coordinates) with the general notion of services'.

Another set of projects focused on mobile objects trajectories rather than individual locations. Jensen et al. [JFCP+01], for example, while adopting the loose definition of *LBS as e-services for mobile objects that involve location information*, enriched the concept and functionality of location information by adding speed and velocity to position of mobile objects. In a data-warehousing perspective, their method makes it easier to analyze and derive the user's activity from their interactions with the LBS. Other related work including tracking trajectory and location-based continuous queries [GBE+00] [ZZP+03] [TFPL04] [XS05] are of great value on predicting and evaluating the services provision. They are especially useful in route navigation [TMK+06] where, for example, knowing the direction of user's trajectory allows avoiding unreasonable answers (e.g. on the highway, refilling ahead is preferred to driving back).

The concern for personalization appears in some systems, such as Hippie [OSJ99], CRUMPET [SBNPZ02], COMPASS [vSPK04] and TIP [HV03]. In these systems, users may specify their interests and profiles. As indicated in TIP, their LBS system enabled to *deliver various types of information to mobile devices, based on location, time, profile of end users, and their 'history', i.e. their accumulated knowledge*. These LBS came up with more attention on the importance of user profiles and facilities to express users' own preferences for personalized information delivery.

Thanks to the miniaturization of electronics technologies and the proliferation of sensor networking [HSK04], a variety of context in different abstractions became available to LBS. Thus, the location is no longer the only context available to affect the discovery and delivery of right information-services [SBG99]. Nowadays, *LBS emphasize the ability to take into account the spatial, time, and contextual characteristics of their interactions with the users to provide the most appropriate services based on the local environment* [YSCA04]. Hence, the functionality of LBS is implicitly progressing from location-dependency towards context-awareness and personalization.

1.4.2 Our Vision of Location-based Services

We see Location-based Services as information services characterized by the following features:

- They are generic information services, not tailored for a specific application (e.g. path finding, road navigation, hotel and restaurant listings).

- They aim at providing information about objects and facts pertaining to a geographic sphere that surrounds the current user's location. The size of the sphere is roughly defined as focusing on the places the user can reach in reasonable time within the same day using local public transportation facilities. For example, for a user in Lausanne the relevant geographic sphere will focus on the city itself and its immediate surroundings (i.e. detailed information about places within this range is expected to be available) but will also include information about Geneva and the nearby cantons (e.g. Vaud, Valais, Neuchatel, Fribourg and possibly Bern) as these may well be the target of a day excursion. The larger area may be described by less detailed information and is consequently most likely to hold the information the moving user wants to retrieve in order to decide about his/her behaviours in the short term (within the coming hours or days).

- They use decentralized knowledge management techniques. LBS in our opinion should not be designed to hold and maintain a centralized database. This is known to have many disadvantages when, as is the case for LBS, information is not the property of the service using it. We expect information in LBS to be gathered from one or more local data sources (e.g., databases and web pages from local tourist offices, local institutions, local domain specific information providers) independently managed by the respective owners. Mediators and wrappers (i.e. services to homogeneously understand different data abstractions and formats), typical of heterogeneous distributed knowledge management, do belong to the set of techniques LBS rely on, and provide service descriptions in different abstractions. As a whole, LBS can be seen as mediators between a generally unknown, usually mobile user and heterogeneous data sources that may have to be dynamically discovered. Thus, LBS contrast with the centralized data approach in mobile yellow-page services, while both aim at providing local information.
- They are primarily geared towards a push mode of operation. They deliver information in response to user's requests rather than in an unsolicited way.
- They rely on positioning services to capture the current location of the moving user (if not explicitly given by the user herself) and similarly rely on the system clock to capture the current time when they are invoked with a request.
- They have to be able to rapidly answer user requests, before the user decides to abandon the request because of the waiting time she endures. This entails that available knowledge has to be screened "on the fly", i.e. avoid techniques that rely on static analysis and preparation of meta-knowledge.

The above features are those we associate with current LBS. As we focus hereinafter on the next generation of LBS we add one more feature we already mentioned:

- They are capable of elaborating personalized and context-aware information. To achieve personalization, they have to exploit user profiles, which contain information about the user. To provide context-dependent information, they need to know about external (i.e. not user-dependent) contexts that may be relevant to elaborate a more precise formulation of the query on hand (the same requests from the same user in different contexts could have different meanings), as well as data contexts that may characterize the available data and allow filtering it on the basis of the combination of current user and external contexts. In these LBS, service matching between user requirements and data service providers is not only based on the service functionalities and spatial and temporal constraints, but also on user profiles and other contexts relevant to user's current request.

We call knowledgeable LBS those LBS equipped with personalization and context-awareness. They are defined as follows:

Knowledgeable location-based services are services able to allow mobile users to specify their requests and profiles on the move and provide them with context-aware and personalized local information relevant to their current activity and request.

1.4.3 Challenges for Location-based Services

The design and development of location-based services faces a wide variety of challenges, some generic to mobile information services some specific to LBS:

Supporting Infrastructure Building the technical infrastructure for LBS requires adopting or developing solutions for many concrete issues regarding positioning technologies, wireless communication, sensor networks, and peer-to-peer computing. A given positioning technology, for example, determines the precision of the coordinates it provides. Functionality and performance in these domains continue to evolve, raising the challenge to design LBS independently of the current state of technology while being able to benefit from the latest advances. New software solutions also have to be invented, e.g. continuous queries processing. All these technological aspects are beyond the scope of this thesis.

User Interfaces This is an issue for all services addressing users interacting with their cellphone, PDA or similar. It groups physical issues related to how to intelligibly display information given the constraints of the device such as limited screen size, limited communication bandwidth, limited storage memory, little or no caching facility, and so on. It also includes software issues such as how to organize information delivery given limitations on the quantity of information that can be displayed at a time, and which paradigm to use in interacting with the user: keywords-based, menus, natural language, limited natural language, command-driven, etc. We do not address user interface issues in this thesis other than assuming that the user request are formulated as a set of structured keywords expressing the "what, where, when, what-else" of the request. The what specifies what information is being requested, the where and when specify the spatio-temporal constraints that have to be used to filter information, and the other is a set of predicates of the form "keyword = value".

Knowledge Representation Spatial and temporal information are intrinsic components of LBS knowledge management. Consequently, LBS have to be equipped with a data modelling paradigm that allows representing this type of information, e.g. where a museum is located (geographically speaking, not just the address, otherwise distance computations, for example, would not be possible) and when it is open. The data model has also to support spatio-temporal modelling in order to be able to take into account the dynamics of users' movements (i.e., their trajectory) and more generically any aggregated variability of values in space and time. For example, assume user Shirley is driving at 10pm on the highway to Geneva Airport. She would like to refill her car before arrival. In such situation, LBS would seek gasoline services near the highway considering her current location, driving direction, and open time and location of gasoline stations. The information services the LBS relies on can also be often regarded as spatio-temporally constrained. For example, a pizza delivery service is formally represented as a time-variant polygon, a bus line is represented as a time-variant poly-line, a gasoline station is represented as a time-variant point.

Data Management As already stated, LBS knowledge is intended to focus on a limited geographical region. It would be easy to centralize all data into a single database. But this information is highly volatile and its regular updating is the responsibility of the information providers, not of the LBS that simply plays a role of user versus the providers. Moreover, access services to these data sources deliver data in their own format and with their own semantics. Distribution and heterogeneity issues can be addressed using a peer-to-peer paradigm. Each information source or service can be considered as an autonomous peer. Query processing would then require a series of operations on services, using service descriptions to perform the searching, composition and orchestration of the relevant services. Although these operations are currently investigated in semantic web research, they cannot be done automatically yet. Therefore this thesis assumes a more traditional distribute data paradigm, where a centrally controlled repository (called data profile in the sequel) holds a description of the knowledge in available local services and its semantics. Their semantics contains the functionalities and spatio-temporal constraints of the underlying or encapsulating services. Nevertheless, we assume that an LBS can communicate with another LBS in a peer-to-peer mode whenever it cannot find the information requested by the user. Our query processing strategy is discussed in chapter 8.

Personalization Personalization is a desirable feature for almost all information services, with possible exceptions motivated by privacy concerns. Yet it is difficult to achieve, as it requires appropriate knowledge about the user. Many web services prompt users to provide their profile by answering a questionnaire. Such a static and rigid solution can hardly be the approach for LBS interactions with mobile users. Application-specific solutions are also plentiful, but LBS are not application-specific. Other approaches rely on regular interactions with the same user to gradually learn a profile for this user. A LBS may not have a history of interactions to build on. In short, many questions are still looking for an answer, e.g.: What should the content of a user profile be? How should it be organized and described? What are the rules to use it and to maintain it?

Context Awareness Would you be satisfied if a LBS suggests you to take the 5.44pm train to get to the airport, while this train does not run on week-ends and you are leaving during the week-end? What if the suggestion is to reach the airport by car, which usually takes 40 minutes, without noticing that the day of your trip there is the automobile show in the building next to the airport, which entails high probability of heavy traffic jams? These are examples of the problems the user may face by user a context-unaware LBS. Unfortunately, well established rules for context description, acquisition, use and maintenance remain to be invented, despite the many efforts that have addressed context definition and management. Similarly for how to formally represent and apply the contexts and their dependencies (i.e. dependencies between contexts and services or between contexts and user profiles) in semantic information matching. How to select the most relevant context for a given query and how to automatically detect a switch in context from one interaction to the next, if possible, are other challenges. These are semantic issues, more complex than, for example, choosing and switching interfaces/channels in mobile HCI.

1.5 Standardization initiatives relevant to LBS

The shift of Internet business modes from B2B to B2C entails that customers more actively than ever involve themselves in the business transactions as the seller, buyer, or the third-party. Such blurring of the distinction between customers and providers has contributed to the emergence of the concept of *service* as a generic abstraction. Meanwhile, the popularity of the WWW and the maturity of wireless technologies opened up more opportunities for the advancement of Web services. The latest draft of W3C defines *Web Service* as follows:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL[5]*). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

To allow services to interact or interoperate without human's intervention, Web services activity actually entails a series of initiatives and standards. As shown in the definition above, the most elementary ones cover the common syntax XML, language description (i.e. WSDL), the format and processing rules of messages (i.e. SOAP[6]), the registry-like directory (i.e. UDDI[7]). In addition, web services encompass other issues, such as security, versioning, and multiple interfaces. All of this is certainly relevant to LBS developers.

The term 'Semantic web', coined by Tim Berners-Lee, describes a dreaming vision where the meanings of web sources can be made explicit in a language which facilitates the searching and integration of web information. Thus, how to efficiently add and manipulate semantics to web service descriptions has been the gist to achieve semantic web services. In our view of LBS, service descriptions play an important role for the definition of what we term the data profile.

Recently, knowledge description standards for the semantic Web have been proposed and are still being refined. Among, the pioneering ones are OWL-S[8] and WSMO[9]. OWL-S attempts to separate semantic matching from aspects of profile modeling, process modeling and grounding. In WSMO, the basic idea is to express the web services and goals as different ontologies and these ontologies can support semantic interoperation using different mediators. Ontologies definitely contribute to the functionality we expect from knowledgeable LBS.

Another initiative that obviously relates to the LBS world is GML, the Geography Markup Language specified by Open Geospatial Consortium for the description of spatial and temporal data. It provides more

[5]WSDL: acronym of Web Services Description Language, candidate recommendation proposed by W3C. For details, please refer to http://www.w3.org/2002/ws/desc/.

[6]SOAP: acronym of Simple Object Access Protocol, proposed by W3C. For details, please refer to http://www.w3.org/TR/soap/.

[7]UDDI: Repository for Universal Description, Discovery and Integration technical reference information. For detailed specification, refer to http://www.uddi.org/specification.html.

[8]OWL-S is an OWL-based Web service ontology. http://www.w3.org/Submission/OWL-S/.

[9]WSMO: acronym of Web Service Modeling Ontology. For details, refer to http://www.w3.org/Submission/WSMO/.

comprehensive and formalized representations on geospatial phenomena, observations and values. Especially, it explicitly specifies the spatial and temporal properties, domains and units of the schema in XML and supports user-customized data-type. It has come to being a practical geospatial reference for geographical web services and support for spatial data integration.

1.6 Thesis Focus

Consistently with the research directions of our laboratory, LBD, this thesis focuses on semantics-related aspects in LBS. The main contributions of this thesis are expected in following aspects:

- To propose a semantic architecture for knowledgeable LBS. By semantic architecture we mean that we identify functionalities rather than software components (although in a given implementation each functionality could be assigned to a separate software component). The purpose of the proposal is to give LBS designers and application designers a better understanding on what functionalities LBS should provide to mobile users and how to apply other information (such as context and profiles) to improve LBS services.

- To identify context-related functionalities in LBS and propose a conceptual modeling approach on managing contexts and supporting context-awareness in LBS. The goal is to apply relevant context information to provide right information at right time and right place.

- To define a conceptual model for user profiling in LBS to facilitate mobile users to specify their profiles and to customise the information delivery on the move.

- To propose an ontology-assisted query processing strategy based on a service-oriented paradigm and allowing for heterogeneity in services descriptions. The proposed strategy shall efficiently perform searching and semantic matching considering user requests, contexts, user profiles, and data profiles.

1.7 Thesis Outline

The thesis is composed of nine chapters. The remaining parts of the thesis are organized as follows:

Chapter 2 presents the *State of Art* of Location-based Services. It sets out with our design of a semantic framework for Location-based Services. The framework mainly consists of seven functional components, which autonomously but cooperatively support user query formulation, context information management, ontological semantic assistance, user profiling, data profiling, syntactic alignment, and query matching. Next, the relevant techniques and applications in literature are investigated and discussed.

Chapter 3 explains how a semantic data infrastructure can efficiently organize and maintain the knowledge in LBS. Characterized by 'Locality, Mobility, Dynamics', LBS calls upon a new strategy in data management. This data infrastructure is mainly composed of a modular core ontology and a set of profiles. Rather than storing or integrating the data instances as in conventional information management, LBS just relates the profiles' data with their corresponding concepts in the modular core ontology. It also introduces the basic elements and features of this data infrastructure, i.e. class, properties, relations etc. In addition, it explains the characters and functionalities of temporal and spatial modules in LBS. Finally, it ends with a discussion of evolution issues in LBS.

Chapter 4 presents the key role in LBS, i.e. data profiles and service profiles. A tree-alike skeleton is the key to structure data services in diverse domains or views with a Is-A taxonomy. In addition, other particular relations, such as *condition, functional similarity, orchestration* are defined for further services' selection and cooperation. Finally, it concentrates on explaining how to align and map the LBS's data (service) profiles originating from various domains or providers to the core ontology.

Chapter 5 discusses an important player in location-based services, i.e. context. Everything is context-dependent. In particular, contexts appear in user profiles, data profiles, even and the user query. Basically, contexts in LBS is presented with a *composition* hierarchy, i.e. space, time, environment, communication, and socio-cultural context. It also presents the specific constructs between contexts, i.e. synchronized/sequential composition constructs. Finally it addresses the feature of multiple representations of the context.

Chapter 6 explores two aspects of issues: the essential problems of modeling user profiles to provide intelligent and customized services; the links between concepts of diverse modules and their effects in contextualized and personalized query processing. Each user can possess more than one user profile. Each profile may be relevant for one or more contexts. Profiles from different users can be analyzed and clustered together according to their similarity and to context, thus leading to the possible definition of user groups. By identifying the objects, relationships, and dependencies between properties of a profile at different abstraction levels and granularities, a conceptual modeling approach of user profiles is proposed.

Chapter 7 defines the query expression and reformulation in LBS. To make it concise and precise, the query is simply expressed with a tuple as <what, when, where, what-else>. It allows users to flexibly express their queries in a free-text format. In addition, the user can further elaborate constraints on thematic target, the spatial range, temporal availability and specific preferences. It provides a unified interface for different data services and the first context information (i.e. location and time) configured or complemented without external hardware setting-up, e.g. sensor networks.

Chapter 8 describes query answering and relaxations in LBS. It starts with presenting the overall query processing algorithm. Context and user profiles pervasively influence the whole process of services matching and personalization. Firstly, deterministic and influential constraints related to the original query are identified, using the relationships with the targeted service class. Correspondingly, the query is rewritten after compromising these hard and soft constraints. By transforming the conjunctive query into one of RDF-compatible query languages, i.e. SPARQL, we show how to match the query against the core ontology and service profiles. When the user asks for additional information or perfect matching can not be achieved, LBS are supposed to offer alternative answers by discovering the misconceptions and relaxing constraints in original query. The relaxation rules for each service class are specified in the corresponding *query relaxation profile*.

Chapter 9 concludes the thesis and addresses what could be improved in future work.

Chapter 2

Foundations of Location-based Services

A model is an abstraction of something for certain purposes, rather than the universe of discourse. This research work addresses modeling issues in designing context-aware and personalized LBS. We attempt to explore three interrelated but different facets of modeling issues in LBS, i.e. user profiles (UP), service data profiles (DP), and context information (CI), and to illustrate how to apply the above-mentioned information, their interrelations and dependencies to enhance the LBS information selection and exchange. Different researches have independently delved into one or more of these realms. However, to our best knowledge, neither other database models nor emerging ontology-based method have proposed within one framework all the modeling and manipulation facilitates that are pursued in our design. In addition, our modeling approach not only provides an abstraction of information involved in LBS per se, but also their proposition may turn to enlightening and boosting the practical modeling ways in other emergent services.

This chapter will present our framework of LBS from a data management viewpoint, and give a representative survey of each player within the infrastructure. In detail, this chapter will investigate different approaches of representing and reasoning context in literature, and up-to-date issues such as manipulating and transforming sensed data to context information. The review will also cover various means of discovering and modeling user preferences, and standards to describe user's preferences and privacy using XML such as CC/PP by W3C. Through our discussion, traditional strategies on service management, such as 'yellow page' are insufficient for distributed and diverse services in LBS. The interface and query formulation in existing applications will be compared and discussed at the end of the chapter.

2.1 Overall Architecture

The design of a new framework must be able to relieve off of the work that preceded it, or be able to resolve the new problems. In this section, firstly we will discuss the related work on the frameworks for LBS from different viewpoints, and then we will present our design and explain the functions of each component and their interactions.

2.1.1 Application-Specific Location-based Services

Through a GPS device, the location service can deal with requests such as 'where am I' for route navigation [Gar04], and 'SOS' in road emergency, or track the target assets in fleet management. Indoor LBS scenarios address requests such as 'find my colleague in the building', as illustrated in EasyLiving [SKB+98], and call-forwarding services as elaborated by Active-Badge [WHFG92], aimed at obtaining the latest target's location with Bluetooth[1] or RFID [HSK04] technologies in a given building. In these systems, LBS infrastructures are relatively simple and mostly oriented towards a single application so that a central database generally suffices for efficient management based on user's ID and location, and the interactions are centrically controlled by the local server.

Apart from traditional turn-by-turn routing (i.e. a routing means like 'at the second street-cross, turn left, and then go ahead until ..., turn right...'), 'where is the nearest restaurant' alike requests increasingly emerge, and call for the convergence of self-positioning technology, advanced location-service support[2], and facilities management. In the telecommunication market, tools are already commercially available that deliver a broad spectrum of content to users' mobile phones according to the location of the subscriber. For instance, in i-area[3] developed by NTT-DoCoMo, the networking base stations can automatically capture the location of the user without explicit specification by users, and the facility information (e.g. map, accommodation, and traffic etc.) to be retrieved about a specific region can be chosen from the menu provided by the system. However, its facility management still resorts to a centralized strategy (i.e. a local or cellar-based database), and its interactions are carried out within a simple and universal interface, i.e. a set of pre-defined menus. It is because the variety of users' software systems and the diversity of applications that the application-independence LBS in true B2C mode is still faraway from the market. We will continue to discuss the complex LBS with multi-applications in next section.

2.1.2 Diverse Frameworks for LBS

Not only can an LBS provide mobile users with useful and local information, like 'the nearest facility', but it is expected to be able to implement sophisticated and divergent mobile services based on detailed knowledge of customer profiles, history, needs, and preferences [RM03]. Moreover, an LBS gathers information from heterogeneous sources, usually in response to users' requests (pull style), rather than receiving unsolicited (push) information broadcast [SAJY05]. Such an elaborate LBS needs careful design and analysis.

The *GUIDE* system [DCMF99] relies on numerous interconnected wireless cells to deliver nomadic users with relevant tourist information (geography & context sensitive). Departing from centralized storage approaches, the data is locally stored and processed within each cell. The information about the objects (e.g. museum, castle or cafe-room.) in each cell is associated with a set of HTML pages and interrelated according to the location, nearness, and type of the objects. The main disadvantage of GUIDE is that the **broadcast** mechanism does not support user preferences and user-driven filtering.

[1] http://www.bluetooth.com/bluetooth/
[2] advanced location-service support may include a series of services, such as Geocoding/Reverse Geocoding, Map Portal, transportation networks
[3] i-area is a service that automatically selects and displays i-mode content related to the location of the i-mode user.

Middleware-based architectures [Ber96], characterized by application-independence, crossing platforms, dynamicity and evolution, and supporting standards, is a popular option to construct a robust distributed information system with a wide variety of applications across many industries. In [Jac04], by integrating a location & position service, a *middleware model* holds the potential to support the development and deployment of end-to-end LBS. In this framework, LBS middleware is the pivot to interact and offer different kinds of services to the subscriber, network operator, and application provider. In a typical sequence of *pull-service* mode, after looking up user's personal information (i.e. subscription and authentication), identifying relevant location, the LBS invokes the application, and then the application matches against content providers and returns the results to LBS; finally, the LBS responds to user's request. Similar proposals are also embodied in [dIL01], [CDM+03] and commercial products at IBM[4], ORACLE and Microsoft.

The *CRUMPET* project [SBLPZ03] uses standard-based agent technology to enable the fast creation of robust, scalable, seamlessly accessible nomadic services within the tourism domain. Generically, the architecture of Crumpet[5] is featured by the client-mediator-server three-tier approach. Through unique interfaces, both the GIS services and distributed service providers can set up the transparent communications with brokers using different and proprietary protocols. In particular, this project applies and influences a series of standards, such as networking standards (i.e. ETSI GPRS standards[6] and 3GPP UMTS standards[7]) for communication protocols, FIPA agent standards[8] for interoperability, W3C standards for service delivery, and Open Geospatial Consortium's standards for spatial information alignment.

Other LBS designers rely on *event-based systems*, emphasizing the impact of external events on the relevance of information delivered to the mobile users. For instance, [HV03] combines an event notification system with LBS to deliver history-accumulated information to users of the LBS.

Different LBS have different strategies to couple location information with location-based applications. The AROUND [JMRD03] architecture allows applications to freely select services associated with their current location. It is mainly composed of three models, proximity model, scope model and functional model. Di Flora et al. [dFFRV05] propose to integrate different positioning technologies in the same LBS to ensure continuously tracking mobile users wherever he/she moves to (i.e. indoor or outdoor). Several recent projects [BJ05] [HL04] [DG03] develop privacy-enhanced LBS infrastructures to cope with sensitive privacy issues.

The data manipulation and modeling approaches in abovementioned works are distinctive in terms of frameworks, protocols and standards. Data-related features in LBS, i.e. multi-dimensions, multi-resolution, imprecision and continuous changes, have been discussed in [JFCP+01], where the authors focus on a data warehouse-based management approach. In [Jen04], the author intensively discusses the database aspects of LBS, especially, the objects modeling issues in LBS, i.e. transportation infrastructure, from spatial data modeling to data update and caching. [TP05] distinguishes three types of data, i.e. domain data, content

[4] LBS at IBM, *Location-based services-wherever you are, wherever you go, get the information you want to know*, http://www-128.ibm.com/developerworks/library/i-lbs/.

[5]CRUMPET WP3 - Nomadic Application Support, Public Deliverable 3.3, http://www.elec.qmul.ac.uk/crumpet/docs/deliverables/d33.pdf. Document ID: 20147/Sonera/DS/D33/A1.

[6] http://portal.etsi.org/Portal_Common/home.asp

[7]http://www.umtsworld.com/technology/overview.htm

[8] FIPA standards for interoperability among software agent platforms. www.fipa.org/

data and application data, and models these data in UML, using ontologies to enhance the semantics and expression power.

2.1.3 Our Framework

This section describes our infrastructure of LBS, which differs from previous work by our emphasis on modeling and exploiting the data and their semantic interrelationships within the LBS (i.e. context, user profiles, and data sources). Regarding the positioning issue, we abstract from adopting a specific solution, and just assume a location-service is inherently embedded in the LBS and holds the knowledge about the location and possibly the movements of mobile users, as well as location and available coverage of location-based applications.

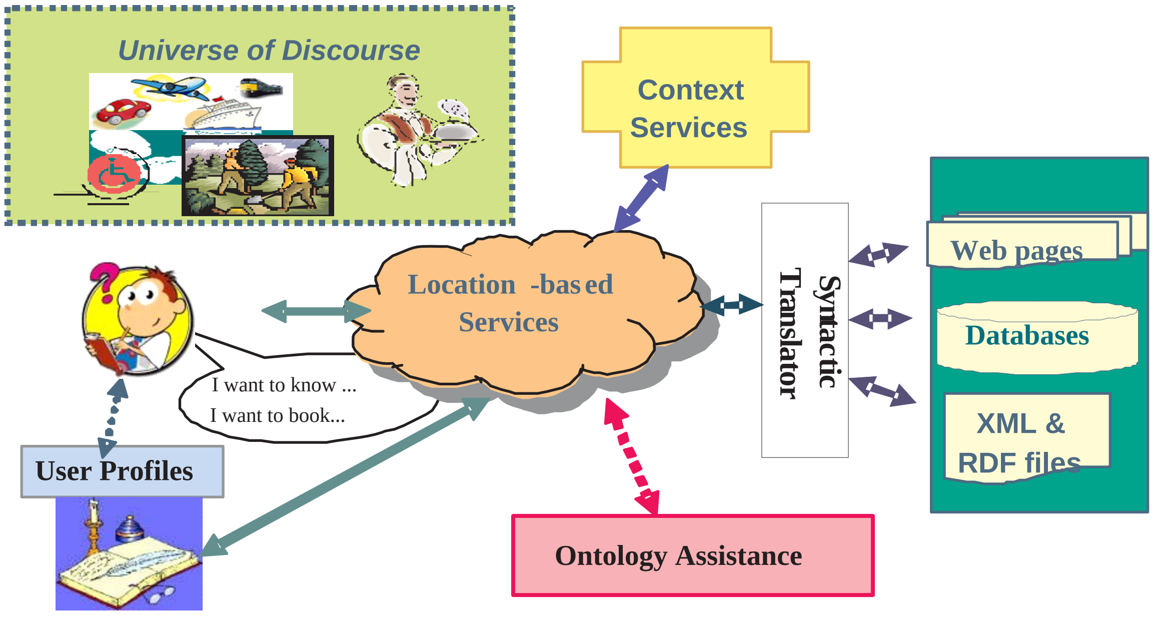


Figure 2.1: An illustrative Framework for Location-based Services.

As shown in Figure 2.1, our framework contains the following components:

- **User's mobile device**

  Basically, a usually handheld device functions as the visual interface between the user and the LBS for information (and service) request and further interaction. Most devices enable to position the mobile user (e.g. through GSM or GPS) and cache the recent requests. In addition, via the device, the user can specify and manipulate his/her profiles and privacies locally (in the device or user's database) and keep the consistency globally (with the LBS' servers). As discussed in [KS05], due to different software systems operated on different devices and the distinctive capabilities of devices, to deliver application-independent service is an uneasy task not only for LBS, but for all mobile services. Hence, the standards on mobile devices[9] that aim to unifying the device's properties representing capabilities, configurations,

[9]Device Independence (DI): Access to a Unified Web from Any Device in Any Context by Anyone, refer to http://www.w3.org/2001/di/.

user preferences[10] and environmental conditions has become one of the important initiatives within the W3C community.

- **User Profiles**

  Each user has one or several profiles [YAJS05]. A user profile generically contains three categories of information: factual information (e.g. age, language and education), preferences and privacy specification. User profiles may change and evolve as the context changes. They can be explicitly specified by the user and kept in his/her personal database locally; alternatively, the local version can be used to update user profiles maintained at LBS server side. In our work, we propose a modeling approach for representing the user profiles and enabling the LBS to understand and apply them for personalized information delivery. User profile issues are further discussed later in this chapter and in more detail in chapter 6 hereinafter.

- **Context Services**

  Context services are those services that are able to provide suitable abstractions of context data so as to make LBS aware of their contextual impact on user's request. Different from the categorization in [vBFA05] [HIR02], we refer to the context data as only the environmental data. For instance, either a local weather broadcast website, a monitoring system in shopping hall, or an event notification system can be regarded as a context services. An LBS has to hold the knowledge on the relevance and relationship between the context and other components (i.e. applications and user profiles) for a given request. At the same time, a context service can serve as context provision and information provision for LBS. For instance, when a user asks for the *local weather in the afternoon*, the weather broadcast website naturally becomes direct information source of LBS.

- **Ontology Assistance**

  As suggested in [UG04], the applications of ontology can be functionally classified into four categories in terms of terminology authoring, common access to information, ontology-based specification, and ontology-based search. In our LBS, ontology has multiple facets and specific functions. At a lower level, it can be used as a dictionary to provide linguistic translations. More interestingly, it provides "*a formal, explicit specification of a shared conceptualization*" [Gru93] for each term. In our framework the *ontology assistance* component provides access to a set of ontologies, defined by the LBS itself or imported from other sources to cover different functionalities. Besides the above functionality, the ontology assistance component also aims at facilitating the mediation between different ontologies, e.g. by adding context of ontology using *C-OWL*, and at addressing syntactic translation issues between different ontology languages, e.g. between WSMO and OWL.

- **Syntactic Translator**

  A translator is a necessary part to ensure LBS ability to syntactically understand the categories and functions of the location-based sources whenever a new one is discovered or joins the LBS. The potential

[10]Refer to CC/PP: Composite Capabilities/Preference Profiles, http://www.w3.org/Mobile/CCPP/.

is high that location-based applications be represented in diverse syntactic formats, e.g. Database Schema, Data Warehouse, XML file, or Webpage.

- **Data Services**

  Each of them is an independent and autonomous data source or application, closely associated with the LBS. They allow LBS either to directly manipulate their data (i.e. query their data source) or access to their unique interface for further operations (e.g. booking). Each data service is associated with one or more concepts in the data profile of the LBS, supporting specific functions and meanings.

- **Location-based Service Core**

  There are several internal components residing in the core of an LBS. First of all, the data management base of an LBS is the core ontology, since it enables LBS to understand what data or application is available and where, analogous to the schema for the database. We will give the details of the LBS's core in the next chapter.

2.1.4 Summary

In this section we investigated the existing work ranging from the basis and prototype of LBS, i.e. location services to the typical conceptual frameworks of LBS. As we observed, several approaches may be distinguished from each other, such as the *application-specific client-server modeling*, *middleware-based modeling*, *agent-based modeling*, and *ontology-based modeling*, since the architecture of LBS is greatly motivated by different aspects of factors, such as the functions, the end-users, the domain, the modal/channel, the networking etc. We proposed our infrastructure of LBS aiming at the conceptual level. All components and their functionalities relevant to LBS have been identified. The following sections present a state of art review for the different components of the proposed framework.

2.2 Context Information Support

2.2.1 Definitions of Context

The Merriam Webster dictionary proposes two definitions for the term Context. The first one reads "*the parts of a discourse that surround a word or passage and can throw light on its meaning*". The second one is "*the interrelated conditions in which something exists or occurs (synonym: Environment, Setting)*". The first definition is close to linguistics and dialogue studies, while the second one has a very generic scope and is close to the common-sense of 'context' in daily life. Both definitions are relevant to LBS. The former can be referred to the first task of an LBS, that is to improve its understanding of the meaning of a query submitted by the user. Queries in LBS are limited to as few terms as possible to make it realistic to be entered by the user on her device. It is therefore important for the LBS to explore the "hidden parts" that would have been formulated if using a full natural language interface and will throw more light on the meaning of the query. Similarly, the second definition may be referred to the additional information, external to the query, that can more precisely determine what the user is looking for and generate additional predicates to get more relevant data.

In the computer science community the term context was initially introduced and formalized in the late 1980's to deal with the problem of generality in artificial intelligence, and a formal language was provided to represent and reason with multiple contexts and express lifting axioms [MB98]. With the recent development of ubiquitous information delivery via the web and its further emphasis in addressing mobile users the idea of using contexts to increase information relevance has emerged as a necessary condition for user satisfaction [Dou04]. Evidence of the importance of context can be found e.g. in the call for proposals to be issued by the European community for its FP7 programme. In that document, the terms context and context-awareness appear in most of the proposed areas where innovative project proposals are sought.

Several papers define and exploit different notions of context (e.g., [SAW94], [DA99] and [Dou04]). A frequently quoted definition states that "*context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*" [Dey01]. On the other hand, a formal characterization of context is still missing, and existing research on context-aware services usually relies on ad-hoc and application-dependent definitions of what constitutes context data.

Restricting the scope to LBS services, context can be more specifically defined as any information which may determine or influence the selection of information to be returned to the user in response to a given query. This includes information that may lead to a more focused interpretation of a query. To avoid any possible confusion, in our LBS framework we further restrict the definition of context as only referring to information that describes the surrounding environment but not the user or the data in the data stores (i.e., context data is both user-independent and data-independent). Typical examples for such context information include atmospheric data, traffic conditions, calendar data (including national and local holidays), and cultural settings. We separate the user-centric context information [vBFA05] from our notion of context and fuse it into user profiles, and similarly define the data profile to hold characterizations of data available from the data providers.

2.2.2 Context-aware services

According to [HIR02], context is classified as sensed, static, profiled and derived context. *Sensed context* refers to the context obtained from some sensors, e.g. the user's location is one of sensed context; *Static context* generally remains fixed and have high confidence, e.g. the device type and its channels; *Derived Context* is calculated using derivation functions and known context, e.g. with the locations of two objects, to judge if they are close to; *Profiled context* is context directly provided by user, e.g. the agenda. From the application's viewpoint, context data can be either input by the user and the system developer, or be acquired by the sensors [ZG04].

In the earlier work, e.g. CyberGuide [AAH$^+$97], the context data is sensed and supplied for certain application(s) in a tight-coupling fashion. This leads to the difficulty for software engineering practices to implement goals such as reusability and multi-abstractions of context. For instance, two applications may require location information of certain product in different resolutions (coarser: in the building; finer: precise location). Dey et al. [DSA01] proposed a conceptual framework and corresponding building blocks (i.e. context widgets, context interpreters, context aggregators and context services) to provide an efficient way to design and implement context-aware applications, to realize context reusability and facilitate application evolution. Similarly, in project [LSI+02] conducted at IBM, a middleware-based context service infrastructure is put forward to support context acquisition and dissemination for context-aware applications. These frameworks allow the context-customers to directly obtain and operate on the high-level abstractions of context data, rather than dealing with low-level data management issues, such as how to collect, clean and aggregate the sensed context data [ZG04].

We can envision that in future ubiquitous services, context services based on loose-coupling techniques will be highly appreciated. Our LBS framework assumes we can directly use the context data delivered by the context services within the LBS. We rather focus on defining how to relate the context abstractions to other data (such as profiles and data services), and how their relevance affects the query answering process [YAJS05].

2.2.3 Context modeling in Context-aware Services

Context data is distinct from other traditional data by many features, such as ephemerality, uncertainty, imprecision, heterogeneity, multi-representation, and spatio-temporal dynamicity [GS01] [HIR02] [vBFA05] [Lei05]. In addition, there exist intricate interrelations between contexts themselves, e.g. the weather context can influence the traffic context but usually not the other way round. Moreover, for a given task only certain context(s) may be relevant and have an impact on the selection and interactions of data and services, even triggering certain actions. As listed in [Dey01], a context-aware application can support one or more features as follows: 1) presentation of information and service to a user, 2) automatic execution of a service for a user, and 3) tagging of context to information to support later retrieval. Hence, context modeling calls for a powerful methodology, largely different from conventional ones, e.g. ER and object-oriented modeling, and rule-based systems.

As already stated, earlier context modeling approaches are relatively simple, since most of them are restricted to a specific environment and use given types of sensors. For example, the work [STW93] proposes a scalable and pairwise-alike context modeling approach, and their basic sensed context information is location. Context is generally described by a set of variables and their values.

Grey and Salber [GS01] present the context data in terms of the subject of sensed context, relevant quality attribute plus properties of the sensors that provide the information, and illustrate how to apply them to design and develop context-aware applications. Their work provides a baseline for further exploration on context modeling approaches, but is still constrained within the meta-information and lacks formality and generality.

Henricksen et al. [HIR02] elaborate a conceptual modeling approach extending from ORM. Their model, alike in object-modeling, includes similar specifications of entity types and attribute types, but adds a new category of associations and their dependencies to capture some essential features and relationships of context information, i.e. its static, dynamic, sensed, derived and profiled nature. Furthermore, in [HI04], the authors focus on their experiences on the imperfection of context in context-aware software design, and identify the information quality arising from the imperfections as a special entity *Certainty* in their model.

An OWL-encoded context ontology (CONON) is proposed in [WGZP04] to support interoperability of and logic-based reasoning for context information. Relying on an upper context ontology, different semantics of the same context concept can be captured and applied in diverse domains. Their reasoning involves two categories: ontology reasoning and user-defined reasoning.

2.2.4 Summary

Context and context-awareness is becoming a pivotal player in LBS. Since it is fundamentally different from the conventional types of data, and has crucial impact on selection and interaction of services in the LBS, there is a demanding call upon a novel approach to represent and model the context data. By surveying previous work on context data abstraction at the low-level and modeling at the high-level, we offered a basic vision on the existing methodologies on context modeling. Our work focuses on how to relate the appropriate abstractions of context information to the information services and user profiles for a given query, and how their relevance determines the selection of data services. Our definitions of context and of their relevance for service selection will be further discussed in chapter 5.

2.3 Ontology Assistance

Originally, *ontology* is a philosophy discipline. In computer science, ontology has become an artifact, whose most commonly quoted definition is the one given by [Gru93] "*An ontology is an explicit specification of a conceptualization*". In [Gua97], the author introduces a distinction between top-ontologies, holding the inherent semantics of its concepts, and domain, task and application ontologies, whose concept meaning definitions are intended for a specific domain, task and application. Guarino also points out the difference between generic knowledge-bases and ontologies, seen as a particular kind of knowledge bases that respond to the purpose of materializing an agreement by a community of users on a shared meaning of its vocabulary. The recent literature is rich in contributions on ontology engineering, ontology infrastructures, ontology representation and reasoning, and ontology applications (see e.g. [GPFLCG04] [SS04b]).

LBS operate in a dynamic framework characterized by a multiplicity of different and very volatile partners with heterogeneous views of the world. This entails the need for the LBS to elaborate its own view, i.e. its own ontology denoted as the LBS ontology (LBSO) to provide a stable reasoning base for its services. LBSO is mainly driven by data providers specifications (which basically delimit the domain to be covered) and local knowledge (e.g. local naming and cultural conventions). Its initial definition is due to evolve according to user queries, and users or providers satisfaction. Users in particular introduce a high potential for heterogeneity in the concepts used and in the terms to denote them. Unless they are constrained into predefined static analysis of their requirements as materialized by menu-based interfaces, users raise ontological issues (to determine the meaning of their queries) which call for various forms of ontological assistance:

- **Dictionary Support**

  To express the right meaning, we need to choose meaningful words [Mil95]. Terminological knowledge is a necessary component of any information system, and LBS in particular, providing support to users in their need to articulate meaningful requests, be it by inputting keywords or choosing items from available lists. The obvious goal is meaningful interactions, where user input can be understood by the LBS, and accordingly the LBS output can be understood by the user. This is not always easily achieved, especially for LBS, due to the mobile user's unfamiliarity with the local knowledge the local possibly implicit rules to express something. Abridged words and acronyms, for example, are often used to refer to something, somewhere, or somebody. They are indeed particularly convenient and expressive in LBS interactions due to the limited display capability of mobile devices [BGMP00]. Terminological ontologies can provide dictionary-like support for lexicographic look-up within the LBS.

- **Data Interoperability**

  LBS are naturally expected to conform to the principles of semantic web services, where the data sources and services in general are assumed to come with self-explained and machine-readable specifications [BLHL01]. Ontologies are the knowledge infrastructure of semantic web services. Ontologies in LBS allow the diverse data sources to interoperate despite different formats and representations in terms of abstractions, similar to the 'Common access to information' in [UG04]. Today LBS can already use any of the ontologies developed by the W3C community and made available via a public ontology library covering a broad spectrum of fields[11]. These ontologies, whether they are private or public, whether they describe other ontologies or themselves, can definitely help information sharing and reuse within an LBS.

- **Data Mediation**

  This is a typical aspect of data interoperability. Mediation services build bridges between ontologies with different syntactic specifications or in different ontology languages. In the semantic web services community, for instance, syntactic mediators have been proposed for reconciliation between OWL-S and WSMO specifications. This seems especially useful for our LBS infrastructure that allows different formats of service descriptions from service sources.

2.4 Data Profiling

Data profiles describe data services, providing the information about what data and functions a service can offer [YAJS05]. They are similar to database schemas in the fact that they are metadata containers, but play a different role. A database schema defines and enforces the rules that constrain the data to be stored in the database. Data profiles inform the LBS about the nature and semantics of data that is stored by the data providers. The data profiles within an LBS independently stem from heterogeneous, distributed, interrelated

[11] http://www.daml.org/ontologies/

service providers. Centralized data management as in traditional DBMS is not necessarily the best strategy for managing data profiles. Decentralized strategies have also been investigated.

In earlier work on LBS, such as CyberGuide [AAH+97] and Active Badge [WHFG92], the data sets cover the identification of user's badge, latest locations of mobile users, and nearby sources. This relatively focused scope makes a centralized data server adequate for information collection and query answering. In GUIDE [DCMF99], tourism spot information is split based on the GSM cell's coverage and is maintained by the local cell server. As the user is on the move, the user can always communicate with the current cell server to get the information on surrounding attractions. Despite its data distribution, data management in GUIDE resorts to the traditional centralized strategy. The HP CoolTown project [KBM+02] presents the difference and correlations between three interrelated worlds, i.e. 'human cognitive world', 'real world' and 'virtual web world', where the web pages are centrally organized at the server, when the context of nomadic user changes(e.g. location), the content will be tailored for the current context(e.g. open the web-page on the introduction of nearby art).

Departing from database techniques, Jensen et al. [JFCP+01] investigate a data warehousing approach to deal with the multi-dimensionality of LBS data (i.e. location, movement and time) and support the diversity of measures characterizing analytical queries. Their ultimate goal is supporting location-based question answering. In [JKPT04] the authors elaborate a more articulated view on their multi-dimensional data model, by enhancing the usual partial containment rule for dimensions and by extending the algebraic support of multi-dimensional data, in particular for the special requirements on LBS location data, e.g. users' history tracks, data of transportation network.

Middleware-based LBS develop an alternative solution to support the multiplicity of independent, autonomous, and heterogeneous services that make up the basis of any LBS. Through the middleware or mediator(s) [SBNPZ02], these services can communicate with the LBS core to provide users with transparent information. In this case, the individual services can maintain their own data, while the LBS maintains higher abstraction about what data and function the service can have. Middleware LBS are a compromise between the centralized and decentralized approaches.

In the ontology-based COSS project [BPvS+04], each data service is modeled by its type, inputs and outputs, and attributes. Ontologies are used for disambiguating the terms in the system, such the location, time, and context. The AROUND project [JMRD03] implements a geographical organization of the underlying services. Services are organized by spatial criteria and location-based query processing strategies are applied to navigate among services while providing optimization based on spatial constraints.

The DBGlobe project [PPT02] clearly adopts a decentralized management strategy. It proposes an ad-hoc database model to locate correlative data stores and exchange similar information within a specific community. The data is classified into the content data, and profile data (i.e. user profile, device profile and movement profile).

2.5 User Profiling

2.5.1 Functions of User Profiles

User profiles provide information that may help in achieving intelligent and personalized information or services. They have already attracted much attention, in particular from research in artificial intelligence [PPPS03], addressing issues such as acquisition of user profiles, learning from user preferences [PB97] [WIY01], use of profiles for better information filtering and delivery [FD92] [NUR03] [LYM04] and web space navigation [CG00] [RG01]. Different profile-aware filtering algorithms have been implemented in applications such as recommendation systems and web browsing.

Most of the proposals advocating the use of user profiles to improve web services selection are tailored for some specific computing environment or pre-defined application (see, e.g., [SC00]). For example, user profiles are routinely used by web services, many of them asking users to fill a predefined form to register to the service, as in My Yahoo! [MPR00], Monster.com for job searching, and PointCast for news subscription [RD98]. The main goal is to return personalized information to registered users. Another visible effect of users profiles is given by web services that, once they know the country of residence of the user, display their web pages in the language spoken in that country. For example, entering "Switzerland" as country of residence results in getting the next web page in German, although the user may be located in the French-speaking or Italian-speaking parts of Switzerland.

Moreover, user profiles can be of great value to information sharing within the same interest group. To enhance social-awareness, GUIDE allows city visitors to interact and cooperate between system-users in the same geographical context (e.g. area), and is of great benefit to evaluate the popularity of attractions referring to comments from other users [CSM$^+$01]. As interviews with potential users of LBS have shown [Kaa03], user expectations about personalization services focus on privacy control, generic and dynamic user profiling, and the ability to share information with others. Users also express their strong intention to build their personal information related to locations, while they are not willing to separately define profiles for each service and each context of use.

In location-based services, due to the inherent mobility framework, the computing environment is continuously changing, as well as the type and functionality of available data sources. Static approaches are therefore poorly useful. In LBS, both the environment and the user profile may change anytime due to change in user's location, in the social environment (e.g., entering or leaving a meeting), and in the user's activity (e.g., from professional to leisure) [YSCA04]. Hence, LBS rather have to focus on more generic and dynamic techniques, anytime capable of adjusting their services to the current environment and current user profile [SAJY05].

2.5.2 Acquisition of User Profiles

Explicit acquisition of user profiles, e.g. using questionnaires and registration forms as we just mentioned, has the clear advantage that the acquired information is precise and reliable as it is entered and validated by the user herself. Beyond the tediousness inherent to form-filling, the disadvantage of the approach is its poor adaptability to changing contexts. Users are unlikely to repeatedly modify their profiles in order to adapt them to changes in the offered services or in the supporting interfaces and environment [Kaa03]. Users also resent answering a lengthy questionnaire for a simple service. Systems willing to avoid resorting to form-filling interactions use techniques for implicit acquisition of user profiles. For example, it is possible to infer user preferences through the observation of the interactions between the user and system and of the information selection choices done by the user. Alleviating in this or similar ways the burden on users looking for personalizes services has a price, that is sacrificing on the precision and reliability of the inferred profiles. In particular, in multi-context environments such as LBS, it is not easy to correctly determine the impact of a specific context (among the combination of contexts that rules each interaction) on user's decision-making process. One realistic way is the combination of the implicit and explicit acquisition strategies. Additionally, LBS that maintain a data profile (as in our proposed framework) can use it to focus user profile acquisition to what may actually be useful, thus limiting the number of interactions with the users devoted to user profile acquisition. For example, it is obviously useless to ask users which type of cuisine they prefer if the description of restaurants in the data sources does not include the type of cuisine they offer. Matching of the user profiles with the data profile identifies the relevant intersection between the two. User profiles acquisition can be incremental, growing with the number of interactions. It can be made more intelligent by taking into account other criteria such as usability, selectivity of attributes and frequency of use. It may also be enriched by coupling its elaboration with ontological reasoning aiming at deriving complementary data.

Not all projects we mentioned so far consider user profiles. Typically, broadcasting approaches ignore almost by definition who their users are. The GUIDE project, for example, disseminates information merely in a broadcast manner so that there is no possibility for end-users to describe their interests or indifference to selected types of data. The system merely attempts to enhance its social-awareness via information exchanges between individuals with same interest at the same place. The most common approach in existing LBS is to allow the user to build up an individual ontology to express his/her preferences, in which all concepts are chosen from a bird-view of available services, so that the system can provide customized information once the nomadic user is approaching services matching the keywords in her user profile [HV03] [TP05].

Comparatively, CRUMPET [SBNPZ02] goes a few steps further in personalization. Building on its awareness of the location of the mobile user and on its knowledge about user preferences, the system provides users with proactive sighting tips and personalized adaptive maps. In addition, it supports ways to implicitly way to acquire user profiles and allows user to update and override their content. However, the tight coupling between the domain taxonomy and the user profiles challenges the suitability of user profiles once the services evolve. Additionally, other contexts beyond location are apparently not applied in its user profiling process.

2.5.3 Preference Modeling and Operations

Personal preferences can and should influence every aspect in user-system interaction. Unfortunately, preference definition, use and management are still very much open research areas. A good preference model affords a formally defined and generic way to manipulate and maintain preferences. Öztürk et al. [OTV05] define the basic notions in preference modeling, and review different formalism such as fuzzy set and non-classical logics complementary to the classical modeling approach. In the database community, [Kie02] proposes a strict partial order semantics for preferences, which facilitates treating preferences as soft constraints when answering queries, especially in e-commerce applications. [KK02] and [KHFH01] embed these preferences modeling techniques into the SQL and XPATH standards to make them commercially available. More recently, [CCC+04] presented a description-logic based framework for preference modeling and matching against service descriptions.

However, the aforementioned proposals fail to provide adequate modeling techniques for preferences in LBS, since preferences in LBS are context-sensitive. In [HV03] and [STW93] the user profile is simply composed of keywords and/or a set of attribute-value pairs. Since the concepts in the user profile are constrained within the range of the subject hierarchies pre-defined by the system itself, the user profile can not be easily reused and adapted for new applications extending beyond the domain boundaries of the existing system. Tryfona et al. [TP05] consider user profile and device profile data and describe them using ontologies. Each user profile is aggregated with different roles, each of which holds diverse keywords that can be matched against service descriptions to facilitate their automatic selection.

Preferences can change as other contexts change. For example, in a user profile, the favorite sports may be defined as "surfing and diving" in summer, and as "skiing and indoor swimming" in winter. The *favorite sports* item in this profile is *season-sensitive*. To the best of our knowledge, few formal models consider the impact of context on preferences. The model in [HK04] is an ER-based preference model where each context ('*situation*' in their terminology) is modeled as an aggregation of location, time, personal influence (i.e. role and activity) and surrounding influence (e.g. device). Each user has a set of application-specific configurations. Thus, for each user and for each application, a context holds a specific set of attributes. However, it is impossible to reuse existing context knowledge to current application for current user, for instance, context in a similar application for the same user.

2.5.4 Preference Interoperability Standards

Ubiquitous computing environments (e.g. hot spots and wireless services freely made available in public areas and meeting places) have made web services available to mobile users. Current technologies however seem cumbersome and inflexible, their use raising a series of new challenges, such as how the system can understand the features and capabilities of different devices, how to customize the information content presentation to each user's device, how to apply different models to deliver information in diverse contexts, etc. To address such challenges, the WC3 community is focusing on enhancing the standardization on device specification, customizing content selection, authoring web service provision and many other standards contributing to support web access for anyone, anywhere, anytime, and using any device. The typical paradigm beneath these efforts is 'the Ubiquitous Web Domain' paradigm.

One of the earlier W3C initiatives in preference-driven services is CC/PP. It aims at making easier for user agents and web devices to specify their capabilities and preferences to customize the content selection (e.g. resolution) and presentation. Its RDF-based framework also enables user to create vocabularies for expressing device and agent capabilities without insurmountable obstacles in HTTP format. Currently, the work on CC/PP has integrated in 'Device Independence' working group. This group extends the original motivation from CC/PP to focuses more on providing mobile users with access to a unified web from any device in any context. This means it not only takes into account the device's capability and user's preferences as CC/PP, but also encompasses the potential influence of context on web information delivery to a specific request, e.g. concerns of modal, connection, location, environmental and level of discourse nature.

In particular, a series of complementary works are developing specifications to support multi-modality for web interaction in 'Multimodal Interaction Activity' (MMI[12]), to boost delivering real web content to mobile devices in 'Mobile Web Initiative' (MWI[13]) and to apply web technology to help users to access services from telephone with speech and DTMF[14] in 'Voice Browser Activity' (VBA[15]).

In addition, privacy issues are recognized as a prominent but poorly-explored part in web-driven processes. On one hand, activities in P3P[16] made a great contribution towards unification of the underlying concepts and privacy protection for web users and web sites to present their data in a standardized and machine-recognizable way. On the other hand, the user can hold a clear snapshot about how the websites collect and utilize their data when users access to and interact with the web sites. Moreover, it enables users to control the exposure degree of their data to the websites by configuring the browsing schema.

2.5.5 User Profile Summary

User Profiles are a good means for mobile users to describe their personal information, preferences and privacy protection requirements. In this section, we investigate the work on user preferences, in particular, preferences in LBS, context-sensitive preferences, preference acquisition, preference modeling, and standards on preferences in terms of device, modality and privacy. However, there is still open space for further exploring and formalizing the interrelations between preferences and relating the preference to the context in the semantic matching process between context-sensitive preferences and services.

2.6 Interface and Query Formulation

Classical means for searching for information or services include yellow-pages, the Internet, and company experts. When the user is on the move, her mobile device (e.g. mobile phone, GPS) may become the most important and even the unique bridge to access mobile services and information. A practical and easy-operable interface is critical for a successful service design.

Due to the diversity of the knowledge domains to be covered and the complexity of the task, different user-interaction techniques and interfacing styles have been applied in practice, such as keywords-based webpage searching (like Google), forms filling to express database queries, drop-down lists to choose among and within function categories. Google's *Froogle*[17] Service has been recently launched by Google Lab in order to provide

[12]http://www.w3.org/2002/mmi/
[13]http://www.w3.org/Mobile/
[14]Dual-tone multi-frequency, http://en.wikipedia.org/wiki/DTMF
[15]http://www.w3.org/Voice/
[16]Platform for Privacy Preference Project, http://www.w3.org/P3P/
[17]http://froogle.google.com/

a means to combine keywords search with drilling down through categories to help users find products within 15 classes defined in the regular browser. In the SPIRIT project [FJA05], the query is specified as a subject of interest and a geographical location. With the assistance of a structured text interface and a map, the query can be further disambiguated using a domain ontology and a geographical ontology. In some GIS-enabled services, sketch-based querying is prevalent because of its visual intuitiveness.

Gong and Tarasewich [GT04] discuss mobile interface design and build on the ”golden rules” for interface design defined by Shneiderman [SP05] to eventually abstract four important guidelines: 1) Enable frequent users to use shortcuts, 2) Offer informative feedback, 3) Design dialogues to yield closure so as to enable the system to answer all potential questions in the dialogues, and 4) Support internal locus of control according to the knowledge organization in the system. Hence, the gist of mobile interface design is to be intuitive and universal. However, in the above methods, they neglect the importance of the context on interface design.

Whatever interface is used, firstly the goal must be clearly specified. The most popular query expression is keyword-based, such as Google, Yahoo. Other approaches includes the combination of keywords input and category selection, such as Monster.com for job searching. For LBS, location is often the most important condition in a query. Unfortunately, to the best of our knowledge, only few services (e.g. SPIRIT) allow users to explicitly specify spatial constraints in their query expression. Rather than offering real spatial services, most interfaces include spatiality specifications only as a filter on places names to be used in retrieving data from web-pages. Recently, the web services community has proposed WSMO as an environment allowing users to specify their query as a set of goals. However, the limited display capabilities of mobile devices restrict the usability and feasibility of the approach.

2.7 Summary and Future Trends

The semantic web is intended to make more and more information and services available to anybody, anywhere and anytime. As a paradigm of semantic web services, Location-based Services have gradually come into prominence because they hold the potential to revolutionize many fields of our socio-economical life, from environmental monitoring and conservation, to manufacturing and business assets management, to automation in the home appliances and health-care in the accelerating mobile environment. However, their design and implementation remains an open issue. In this chapter, we have defined an overall architecture and provided a review on related work for each component involved in this infrastructure. Until now, there is no adequate design for context-aware and personalized LBS, where different components must consistently interoperate, so it still calls upon the confluence of many emergent techniques, e.g. wireless sensor networks, context-awareness, semantic web, ontologies, user preference modeling, privacy protection, data integration, etc.

Chapter 3

LBS Semantic Data Infrastructure

3.1 Introduction and Motivation

Traditional data management applications operate in well-structured information environments, e.g. in database-centered frameworks, and benefit from full-fledged strategies (e.g. SQL) that provide the needed functionality to represent, manage and query the well-structured data. Web-based application environments, on the other hand, face a cumbersome task trying to provide the same functionality for the huge amount of heterogeneous data resources that the web makes available to occasional as well as regular users. Deprived from the database paradigm enabling standard and formatted description of data structures, web data management has developed new solutions to overcome the poor, unstable and unclear organization of web data. The new trend is characterized by an emphasis on elicitation of data semantics, to improve the chances for correct interpretation of the data by heterogeneous partners (users and agents) that do not adhere a priori to common coordinated behavior. The semantic focus is at the heart of the semantic web to support its organization as a universe of interacting services. In parallel, current LBS have grown as services for users on the move. They usually rely on traditional centralized data management techniques, integrating all the data they need into a single repository and providing all their users with the same services. We foresee that a second generation of LBS will be developed in a way that is more consistent with the semantic web framework, so that these new LBS may be used not only by users on the move, but also via the web, exchanging data and services with any agent within the semantic web. LBS however will retain their specificities, namely:

- **Locality, Mobility and Dynamics.** Today, the role of LBS is to provide information related with the current location of the user. For LBS seen as web services, this can be rephrased saying that LBS provide information about a specific region, in particular information related with the current real or virtual location of the user. To fulfill this capability, LBS build knowledge repositories describing all locally relevant data stored in a limited number of data sources. When the user moves from her/his current place to a remote one, the previous sources and knowledge become useless for upcoming queries from the user. An LBS aiming at being able to follow its users from one place to another needs the capacity to dynamically acquire new sources and build the corresponding new knowledge into its repositories.

Alternatively, LBS can be specialized to only serve a specific region. In this case, it will be up to users to switch from one LBS to another (just like cellphone users today switch from one telecommunication provider to another one). This second approach is simpler, although it requires that standards for LBS be developed to avoid users getting lost because of too high heterogeneity of interactions with LBS from different providers. In this chapter we discuss how LBS become knowledgeable about a specific region. This is basic functionality, whatever the approach.

- **Trading comprehensiveness for rapidity.** LBS are intended to serve people on the move, i.e. they have to provide rapid rather than comprehensive responses to user queries. Well-known centralized management strategies that call for long set up processes and static solutions are not well suited to LBS needs. For example, LBS handling of heterogeneity of autonomous data sources dynamically entering and leaving the scene calls for on the fly and incremental techniques for syntactic and semantic alignment, while these are traditionally time-consuming and cost-expensive tasks. This departs significantly from e.g. data warehousing frameworks that have similar data heterogeneity problems but different data quality and comprehensiveness requirements.

- **Modularity.** LBS obviously have a strong specific focus on spatial and temporal information and related constraints. They need new knowledge extraction techniques to acquire such information from web pages and XML files. For example, common knowledge extraction techniques based on natural language analysis (e.g. text frequency computation) have poor performance on understanding the data in the spatial and temporal dimensions. In addition to space and time challenges, LBS aiming at context-awareness have to be highly sensitive to the current state of affairs when looking for answers to user queries. Indeed, they get ad-hoc requests from any kind of user and requests can be about almost anything that is locally related. They have to develop strong skills for context-awareness if they want to be successful. For the same reasons, they have to care about personalizing services based on knowledge they can acquire about user's characteristics. This multiplicity and diversity of concerns make LBS a complex software whose processes constantly need to adjust to running circumstances. To make this possible while keeping performance we propose hereinafter a modular data architecture, better suited than the usual centralized database assumed in current LBS.

Many different techniques can be used in an LBS to organize, acquire and maintain the knowledge needed by the LBS to provide efficient query answering despite the dynamicity and heterogeneity of the data sources and the variety of queries that users can formulate. A key feature is that while the universe of discourse for user interactions is theoretically infinite (i.e. users can ask queries in any possible knowledge domain), from a practical perspective users' queries most frequently remain within a restricted domain that can be easily characterized. For example, LBS used by tourists typically face a limited number of queries regarding the available facilities for tourists (hotels, restaurants, museums, shows, festivals, etc.) and about usual concerns of travelers (e.g. transports, and itineraries). The relevant items in this universe of discourse can be easily identified by looking at leaflets provided by tourism offices worldwide, leaflets that show strong similarity in the way they structure the information. Consequently, the core piece of knowledge supporting LBS operation can be seen as a domain ontology, e.g. a tourism ontology. It can be derived using existing ontologies in the same domain and tuned towards local features using service descriptions given by data providers. However, what LBS need is not a domain ontology in the traditional sense, i.e. a definition of concepts relevant to a domain and independent of any specific application. LBS are a kind of application, whose goal is providing information from the available sources. Following a service-oriented paradigm (i.e. replacing information with information services), the "domain" of the LBS ontology is the definition of available (or potentially available) services found in the local information sources. Consequently, the LBS ontology is an ontology of services and service usability, rather than an ontology of abstract concepts. Because of this service orientation, the LBS ontology is to be equipped with specific features, such as links between services to show functional equivalence used to plan alternative services or the concept of input property to define knowledge that has to be provided when calling for a specific service. We call *core ontology* the LBS ontology of services. Being an ontology of services does not mean that the scope of the ontology is limited to service description. The LBS also needs knowledge about local features that are not services but provide contextual information that is essential to refine service usability given the current state of the local world. They also need generic knowledge that allows understanding the terms and the semantics of users' queries, as well as understanding the relevant characteristics of users and how they can help in personalizing and contextualizing LBS operations. These additional facets (users, context) enrich the core ontology and LBS quality of service. As a complement to this core LBS ontology, the LBS must be able to access external ontologies for example to find out about services currently unknown to the LBS. Another enhancement to the knowledge in the core ontology is to maintain a rich terminological diversity to cope with the variety of cultural and linguistic habits of a totally open and unpredictable population of users.

This chapter presents the knowledge architecture that organizes the various pieces mentioned above, and how this knowledge is incrementally set up and maintained. More details on the service and context description and the user profile component are given in the following chapters.

3.2 LBS Data Architecture

Let us start with a preliminary remark that introduces our focus on LBS for a specific region. We see the LBS operating according to a mixed paradigm, i.e. primarily centralized but occasionally decentralized [YSCA04]. From the centralization viewpoint, each LBS server holds an integrated view of data sources local to a specific region (e.g. a city and its suburbs). We however assume that when the current LBS server is unable to answer the user query because the local data does not lead to a possible answer, before replying negatively it forwards the query to the geographically neighboring LBS servers. Hence, each LBS server can be regarded as an independent peer, directly connected to the neighboring servers. For instance, let us assume that user Shirley is in Lausanne now and will go to Geneva for a meeting this afternoon, and she would like to check the relevant bus time-table in Geneva in advance. Should the LBS at Lausanne be unable to answer her query, the query will be forwarded to the neighboring LBS servers, e.g. the LBS at Geneva. Thus, our LBS data infrastructure adopts a typical peer-to-peer strategy. However, this thesis only concentrates on semantic data management for a single LBS. Issues about query forwarding and processing between LBS peers are beyond the scope of this dissertation.

Regarding data management at the local server, we do not intend to integrate or align all data sources in a single repository. Instead, LBS only have an abstract view of the data sources (i.e. data profiles, see the definition in Section 4.3). The detailed description of a service as stated by the service provider remains at the data source, while the abstract view records only the main characteristics of the service that are needed to quickly estimate if, given a specific query, the service may be relevant or not. To obtain more detailed information about a service, if needed, the data sources are directly queried to extract the instantiations and other properties of services that may be of interest to the requesting user. Our approach shows two benefits: firstly, data sources can autonomously maintain their data, and LBS are just responsible for suggesting users where to find appropriate data services; secondly, the data sources can protect their data based on their own privacy regulations and can constrain users' access so as to only provide their data if certain conditions are satisfied.

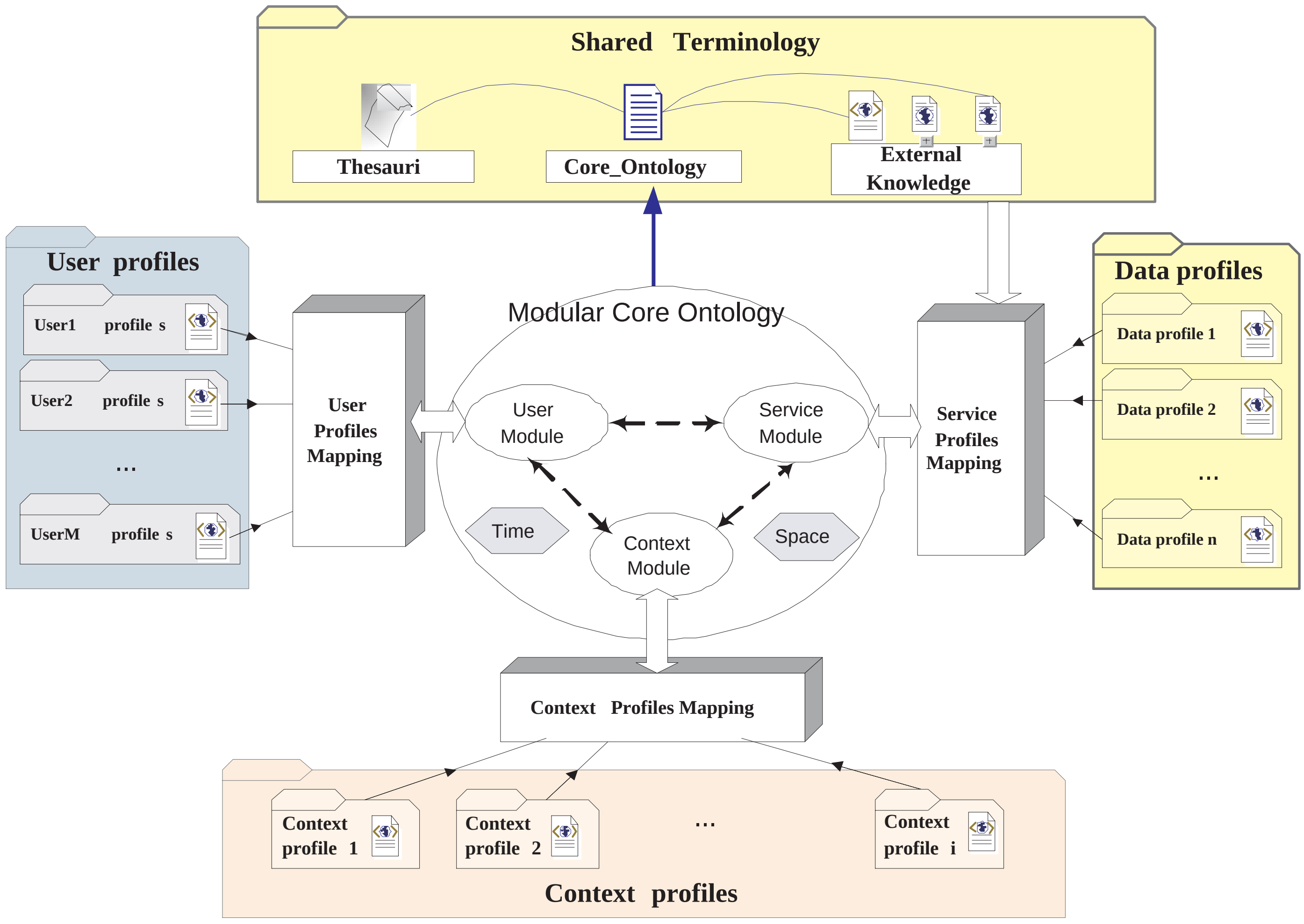


Figure 3.1: The Basic Data Infrastructure in our LBS.

In order to build a knowledgeable LBS system capable of actually relating concepts from different sources, disambiguating the terms in queries, and supporting query matching in multiple dimensions, so as to efficiently respond to user queries, the data infrastructure we propose includes six repositories interconnected via the LBS matching services, as shown in Figure 3.1. Two repositories, called Core Ontology and Shared Terminology, contain the knowledge sub-stratus built by the LBS and its administrators. Three other repositories contain knowledge on the external actors with whom the LBS interacts, users, services, and context providers. We say these repositories contain the user profiles, the data profiles and the context profiles. Finally, a working repository contains the queries processed by the LBS, holding data specific to a given query. For similarity, we say these data are created and maintained as query profiles. The infrastructure components are briefly described hereinafter to give an overall but intelligible view of the infrastructure. Each one will be described later in more depth in a dedicated chapter.

Core Ontology[1]. The core ontology (CO) is the repository for the semantics of the data managed by the LBS. It is the kernel of the LBS's data infrastructure and it is used to structure the diverse aspects of service-related information and perform reasoning about it. It is presented here as an ontology (although it could also be modeled as a database schema) as ontologies are nowadays the predominant approach towards knowledge sharing, reasoning and management in a semantic web environment. It basically describes taxonomies of interest, encompassing a set of definitions of classes, properties, relations, axioms and constraints. These taxonomies organize information in complementary sub-domains. Services, users, context, space and time are the main sub-domain we have identified as essential to LBS. These sub-domains are quite heterogeneous and each one can be pretty complex in itself. Therefore, for better management and improved performance, we propose that the core ontology be a modular ontology composed by one module per sub-domain. A module is defined as a smaller ontology, part of a larger one, which covers a sub-domain within the domain of the larger ontology. This fits perfectly with the LBS core ontology and its multiple taxonomies. Moreover, building modular ontologies is nowadays feasible. Several proposals exist on how to build a modular ontology, how to maintain modules individually as well as maintain the inter-module links that allow interactions between modules, and how to perform distributed reasoning within a modular organization [PSS08]. Modular ontologies have been claimed to solve scalability issues and speed up reasoning, benefits that are for sure also relevant for LBS. However, the primary benefit we see in a modular ontology is its improved understandability and easiness of design and administration. A modular organization allows giving responsibility for each module to an administrator with specific expertise in the sub-domain. This administrator will be able to focus on the sub-domain and will do a much better job than a global administrator in charge of the whole ontology with all its sub-domains. Given that human factors are likely to become the typical bottleneck (or critical cost factor) of future software, modularization seems to be the best way to increase the quality of information in an LBS. Moreover, thinking of the semantic Web and its world of specialized services, we can imagine there will be services specializing in context data, other services specializing in user profiling, and so on. At that point, each module administrator will be able to use these services to elaborate her/his ontology module in the best possible way. Indeed, one of the other benefits expected from the development of modular approaches is improved reusability. If good sub-domain repositories are available, it will much easier to reuse them than to elaborate a brand new one from scratch.

[1]See the formal definition at Definition 3.1

Whether modular or not, reuse is anyway the approach to follow when starting building the core ontology. Initializing the core ontology means inserting all concepts that are assumed to be useful for the targeted application(s). Once identified the targeted knowledge domain, a clever designer will look for existing ontologies in the same or similar domain. If any one is found, its import can form the initial set-up for the core ontology. For example, if a tourist-support application is targeted, the designer will look for, and find, a tourism ontology whose concepts include accommodation, food, transport, and leisure services. Very likely, many of the imported concepts will be generic enough to be suitable for the new LBS and its service module. However, not all of them will be relevant for local use, and not all of them will be formulated in a way that is consistent with local habits. Therefore, in a second step, the core ontology is turned into a *local* domain ontology (e.g. tourism support in the Canton de Vaud, Switzerland). This can be done by acquiring and adding location-specific information (i.e. contextual data), such as local landmarks and local calendars, and enriching existing information, such as adding the preconditions for using a given available service, e.g. to use a motor-boat rental service the user needs to hold a valid sailing certificate. Conversely, making the ontology local also includes removing generic concepts that are locally irrelevant (e.g. downhill skiing for an LBS about The Netherlands).

At this point, the LBS is fully ready to start operation. As long as the LBS is in use, the core ontology is expanded based on the queries received and answers given. Identification of new requirements by the ontology manager will also lead to ontology expansion, but this is very much similar to normal ontology evolution, not specific to LBS, and will not be discussed here. Notice that for the administration of the core ontology given this incremental strategy it is advisable that elements in the ontology be qualified as either prospective or confirmed. A prospective element is one that has been entered in the initialization phase but has not yet been used (up to now). A confirmed element is one that has been actually used during the processing of at least one query. Prospective elements should sooner or later become confirmed elements. Elements that remain prospective elements for too long are candidate for deletion, to be triggered by the ontology managers whenever it is felt that this is a reasonable enhancement to ontology performance (smaller ontologies may be explored and updated faster than large ontologies). More sophisticated maintenance strategies may be defined, based on usage metrics, relevance feedback and other usability criteria. They are not investigated here.

Shared Terminology[2]**.** The shared terminology is a superset of the core ontology that focuses on terminological support to facilitate interoperability among components of the data infrastructure. It aims at figuring out the heterogeneity (from the syntactic level to semantic level) in the vocabulary of data sources, core ontology and user queries. In particular, it functions as a dictionary to disambiguate the words/phrases in LBS. It is helpful for example when a concept in the core ontology is semantically equivalent to, but syntactically different from, a concept in the query or in a data-profile. The shared terminology functions as background information to be used when using the core ontology does not suffice for example to identify a concept that the LBS has found in a query. For instance, it may happen that a French-speaking user formulates a query in French while the core ontology is in English. The LBS, having failed to find the

[2]See the formal definition at Definition 3.10

concept in the core ontology, will look into the shared terminology. The latter may hold a multi-lingual version of the core ontology and therefore enable the LBD to understand the user's terms.

Initially, the shared terminology is populated with the concepts in the core ontology, which we call *internal terms*. Next, each internal term is complemented with its definition(s) extracted from certain thesauri. Subsequently, the shared terminology is further enriched by introducing what we call *external terms*, i.e. terms (from the thesauri) that are somehow relevant to the internal terms, but are absent from the core ontology. *Relevant* means there is a relationship between the internal term and the imported external term. For instance, the internal term 'car rental' may be related to the external term 'hire a car' with a semantic equivalence (synonymy) relationship. During LBS operation any new term appearing in user queries or in service-profiles will be identified thanks to the shared terminology manager and added to it in order to capture the terminologies of users and services that differ from the terminology of the LBS designers. Because the shared terminology manager can use any external ontology to identify the unknown term, we do not expect the identification process to fail. Should this happen, the query or service description using the term cannot be accepted and further human interaction is needed to solve the issue.

Data Profiles. In our LBS data infrastructure, the descriptions of specific services (e.g. a local car rental service) are autonomously created by service providers and are kept and maintained at the corresponding data sources, external to the LBS. These service descriptions, together with a global description of the data source (e.g. owner name, last update date), form what we call a data profile. Data profiles are provided to the LBS by the data source administrators when the data source joins the LBS. The LBS records the new data profile and proceeds to recognize the services it describes (i.e. to understand what services it can provide, what functionality each service has, what spatial region each service covers, when it is available, etc.). Characterization of services is based on their matching with knowledge in the service module in the core ontology.

Service Profile Matcher. This component, SP matcher, is responsible for acquiring data from the data profiles, in particular data from the service descriptions. It holds a set of pre-defined syntactic and semantic rules to transform the heterogeneous data profiles into the format consistent with LBS core ontology. Conversely, it is also responsible for managing the mapping of the core ontology into service descriptions, i.e. the syntactic and semantic support needed to transform users' queries into the format that the data sources can understand. Its first task is to identify the terms in the service profiles and to find out their corresponding terms in the core ontology or in the shared terminology. The process is repeated for the classes, properties, and other features that the service profile encodes: they are identified and associated to the core ontology. The goal is not to fully copy the service profile descriptions into the ontology, but to ensure that the core ontology service module holds enough information about the service to be able to evaluate the effectiveness of the service as a response to users' queries.

User Profiles. The LBS needs to know about its users to implement its personalized services goal. Descriptions of users are traditionally called user profiles. We consider these profiles are stored in a dedicated repository that may be within the LBS or in a dedicated site elsewhere but accessible by the LBS. Moreover, we assume each user profile exists before the user formulates a query. We do not investigate how the user profile is defined and how it is maintained. An LBS could be equipped with machine learning mechanisms to enhance user profiles according to the queries users ask and how they react to the proposed services, but this aspect is beyond the scope of our investigations. The fact that there is no evidence that the same user will repeatedly and frequently enough use the same LBS (a prerequisite for machine learning) is one of the motivations not to address user profile evolution. As for service profiles, users profiles need to be understood by the LBS, i.e. the LBS must identify what the data in the user profile means and how it can be related to context and service data. The LBS user module maintains the concepts generically related to users, possibly abstracted from the actual user profiles.

User Profiles Matcher. This component, UP matcher, has similar role as the SP matcher, but applied to user profiles. It holds similar transformation rules as the SP matcher. It is responsible for acquiring data from the user profiles, i.e. matching the user profile and the user module in the core ontology, in particular the data that is relevant for the processing of a given query. Conversely, it extracts from the user profile data that are relevant for presentation of the query results to the user.

Context Profiles. The LBS obviously needs to know what characterizes locality of information services. Locality includes local factual data, e.g. the description of places of interest within a city, as well as local temporary data, e.g. which events, festivals, etc. are scheduled. Locality also applies to knowledge, such as cultural habits. All of this is used to build the context module, but needs to be regularly refreshed (obvious for temporary data). Context profile is the repository describing the information sources where the context data can be obtained or extracted. It could contain, for example, a set of URLs with associated description of which kind of data is available and how it can be extracted, or a pointer to a tourist office database or set of XML files such as an event schedule.

Context Profiles Matcher. Similarly to the other matchers, this one is responsible for establishing and maintaining a correct connection between the context module and the context profiles.

Query Profiles and Query Relaxation Profiles. Each user query undergoes a multi-step processing by the LBS before its result can be returned to the user. Several steps have to do with reformulation of the original and intermediate versions of the query, for example to take into account user and context information. Consequently, the LBS has to maintain for each query a description of its successive formulations as well as the relevant subsets of the user profile (conveying the user data that has been found relevant for this specific query) and of the context data (conveying the contextual data that has been found relevant for this specific query). These "query profiles" are stored within the LBS as element of a sequence of queries that we call a user interaction. A user interaction represents an exchange between the user and the LBS that leads the user, through a series of questions & answers, to get the desired information. In addition, query relaxation is a very common topic in LBS. It happens whenever the user need additional information or the perfect matching can not be accomplished. For each service class in the core ontology, it corresponds to a query relaxation profile. The relaxation profile contain a set of relaxations rule for each property and its possible values. In addition, a ranking function is also included in the relaxation profile to determine how to select/rank the relaxed query when multiple relaxations are available. Each query relaxation profile is associated with one or multiple query profiles with the same service class. The analysis of query profiles can assist to refine the relaxation rules and ranking functions for better recommendation and more reasonable query relaxation.

Before developing a detailed description of the core ontology and shared terminology, the two repositories internal to the LBS, we briefly show a usage scenario to illustrate the interplay of the different components in the data infrastructure. We first show a set-up scenario, followed by a query scenario.

Set-up Scenario. Let us assume a software company has an LBS skeleton, i.e. all the software to run the planned location-based services, and wants to set-up a first version of its LBS servicing tourists visiting the city of Lausanne. Setting up this specific LBS is a knowledge acquisition process. As we already mentioned, the first task is to find and import one of the existing ontologies for the tourism domain (see, for example, the public ontology at W3C website). The imported concepts will form a first draft for the service module, generically describing standard services in support of traveling tourists (something similar to the organization of yellow pages). The second initialization step is acquisition of the local context. This can be achieved through import of data files acquired from local providers (e.g. the local tourism administration, and tourism-related organizations such as cultural associations, local press, movie distributors, transport companies, and so on). Alternatively, data can be captured from public websites using knowledge extraction techniques (e.g. [TLKT01]). Captured and acquired data are formatted to define and populate the context module. Such a priori knowledge of context may be needed to perform the following step, which is acquisition of service descriptions from local service providers. As stated, we assume that local providers will make their service descriptions available, not necessarily following a fixed format or adhering to a fixed terminology. This is where the LBS will start building the shared terminology, in its attempt to understand what a given service description means. For example, retrieving the service description term from WordNet and associating it to the corresponding term already in the service module. Acquiring service knowledge is a sophisticated task, as the goal is not to import service descriptions but to build the abstract view of services that forms the service module. This knowledge abstraction step relies on linguistic techniques (to identify major relevant terms) as well as on semantic techniques (to only retain what is useful in differentiating the service from the other services). Building the service module obviously includes building the mapping between the module and the data profiles. Once the service and context modules are set up, the LBS is ready to start receiving queries from users. Here the question is whether user profiles will come from the user device or will have to be retrieved from some external repository of user profiles. Given the sensitivity of the issue, the former is likely to prevail. This means the LBS has to run a user profile understanding process, equivalent to the service profile understanding process. Indeed, some generic knowledge about users characteristics can be initialized a priori within the user module. Actual user profiles will be matched against this initial set-up, the matching possibly leading to enriching the user module and the shared terminology.

Query Scenario. How user queries are processed is discussed in detail in chapters 7 and 8. Here we just sketch what happens in the proposed LBS. User queries in our approach are formulated by stating which kind of service the user is interested in, and the spatial (e.g. proximity) and temporal (e.g. current availability) conditions that are to be taken into account while selecting specific services in response to the query. The user query can also specify thematic (non-spatial and non-temporal) conditions to refine the search for services of the given kind. The hypothetical query "give me nearby restaurants still serving Swiss traditional cuisine after 2pm today" illustrates denotation of a kind of service, restaurant, together with the specification of spatial, temporal, and thematic conditions. Query processing within the LBS includes the following phases: 1) understanding the query (are the terms and the service they denote known in the core ontology or in the shared terminology? If not, search external ontologies); 2) retrieving from the service module the prototypical description of this kind of services in order to check preconditions for using services of this kind and check that conditions stated in the query can be actually evaluated (e.g. that the price for the service is available); 3) personalizing the query, i.e. check if relevant knowledge in the user profile allows refining the query, e.g. refining a query for a hotel into a query for a centrally-located hotel; 4) contextualizing the query, i.e. check if context data allows further refinement, e.g. replacing the expression cheap hotel with the expression hotel with price less than 80 CHF; 5) executing the query, i.e. find relevant services in the order of preference, if any, as stated in the reformulated query. Notice that the order in which to execute phases 2, 3, and 4 can be changed without influencing the final result.

We can now move on to a first detailed description of the core ontology and shared terminology.

3.2.1 The Core Ontology

The core ontology is the key component of the LBS data infrastructure. According to the view of LBS as tools providing users with information about services, we can say that the central component of the core ontology is the service module. This module holds the metadata about what services the LBS can offer, how these services are structured (as hierarchies of interrelated classes), and has links to the context and user modules to identify contextual and user-related information that can influence the choice of services in response to users' queries. The hierarchical structure provides a service taxonomy primarily based on a standard classification of services in the domain covered by the LBS. Most likely, a domain taxonomy can be imported from external sources, and then refined (i.e. restricted or extended) to take into account the specificity of locally available services. For instance, it is possible to import a generic tourism ontology and then improve its local relevance by adding more details on ski-services if the region of interest is Switzerland, or deleting ski-services as irrelevant if the region of interest is Hong Kong. Knowledge in the service module is complemented with knowledge that the other modules maintain to support a global apprehension of service data. Spatial and temporal knowledge is obviously needed to be able to describe and reason about the geographical location of services (for queries such as "retrieve the nearest ...") and their potential usability in a given timeframe (for queries such as "retrieve a nearby restaurant that still serves lunch after 2:30 pm"). Space and time modules provide the basic tools for expressing spatial and temporal knowledge. Context and user modules hold the knowledge that is by definition needed to achieve the goal of personalized and contextualized service retrieval. In the sequel of this chapter we describe the generic concepts used to structure the core ontology. The following chapters provide a detailed discussion of the specialized concepts that we advocate for an LBS to materialize the service, context, and user modules.

Following current trends in ontology management, the following description of the content of the core ontology is inspired by the Description Logics (DL) paradigm, the most frequently used paradigm in the ontology research community. More precisely, we refer to OWL-DL because this specific version is likely to be the one on which the Semantic Web will be developed in the short-term future. Known advantages of OWL-DL include: 1) it provides a solid basis for defining the semantics of information, 2) the underlying open world paradigm is well suited for interoperability in a web-based framework, 3) it provides a set of built-in constructs that support more semantics than RDF on class and property, e.g. disjoint, cardinality, property restriction, etc., and 4) its specifications are decidable. DLs, however, are not very reader-friendly, in the sense that their expressiveness in terms of data structures is very limited (equivalent to functional or binary relationship models, as represented by e.g. ORM, the object-role model[3] used in the DOGMA approach to ontologies [JM02]), leading to heavy description mechanics when confronted with complex objects, complex attributes and rich relationships. Another limitation of DLs is their poor support for modeling dimensions, e.g. space and time, that are essential to LBS. Therefore, to improve readability and simplify our discourse, we will sometimes refer to conceptual modeling constructs (taken from the MADS data model developed by our laboratory [PSZ06]) to explain requirements that we associate with the core ontology and are not simply expressible in OWL-DL.

OWL-DL ontologies consist of classes, properties, individuals and their axiomatic definitions. Classes are the nodes of the ontology at the metadata level. Individuals are the nodes at the data level (i.e., instances). Properties are binary connections between classes (object property) or between a class and a value domain (data type property). Accordingly, our LBS core ontology is defined as follows:

DEFINITION 3.1. Core Ontology (CO). *The LBS core ontology is a set of ontological modules. In our approach this set contains the service module, the context module, the user module, the space module and the time module. Each ontological module consists of classes, roles (including object properties and data properties), individuals (or instances), data types, is-a links and the corresponding defining axioms. We use the following notations:*

- *C denotes the set of all classes in the ontology. It groups all classes from the different modules. The set of classes in a module are denoted as: $C^{service} \subseteq C$ is the set of service classes, $C^{space} \subseteq C$ is the set of spatial classes, $C^{time} \subseteq C$ is the set of temporal classes, $C^{context} \subseteq C$ is the set of context classes, and C^{user} is the set of user classes.*

- *I denotes the set of all individuals in the ontology. It splits into two disjoint subsets: I^{class} is the set of class instances, each of which is uniquely identified by its identifier; I^{data} is the set of data values. Data values belong to system data types (integer, string, etc.) or to user-defined data types. For example, landmarks to be found in a specific city are instances of the Landmark class in the context module.*

- *DT denotes the set of user defined data types. It includes the spatial and temporal data types we discuss hereinafter. User-defined data types allow modeling value constructs that go beyond the simple value domains supported by OWL-DL. The current state of art in ontology reasoners does not support user-defined data types, but work is in progress, for example to support complex values.*

- *R denotes the set of roles, i.e. binary relations between classes or between classes and data value domains. In OWL-DL the former are called object properties, while the latter are called data value properties. Roles may be internal to a module, e.g. linking two classes in the service module, or be inter-module, i.e. linking a class in one module to a class or a data value domain in another module.*

[3] Refer to http://www.orm.net/.

Inter-module roles link, for example, service classes to context data on which services may rely. We will talk about roles and role instances to differentiate links at the metadata level from links at the data level.

This chapter focuses on generic constructs for the core ontology that apply to all its modules. Specific constructs are discussed in subsequent chapters. We abstract from the possibility to define sub-modules within a module (e.g. defining a sub-module on products within the service module, describing products available from the known services). The semantics of the modules is self-explanatory. The service module describes all service-related information. It holds information inherent to services (e.g. which kinds of services exist, how they are characterized, what do they offer as product, if any), and holds relevant links to the other modules. Examples of links are link to the space module to spatially locate the services, links to the temporal module to characterize opening hours for the service, links to the context module whenever services are described as sensible to context (e.g. services located nearby a football stadium and selling alcohol may have to close operation some hours before and after a football match), and links to the user module (e.g. for services selling alcohol to state that they need knowledge about user's age to check their accessibility). The service module is discussed in the next chapter.

The spatial classes in C^{space} are the classes that convey the spatial features supported by the ontology software. Spatial features are usually defined to include point, line, surface, polygon, etc. They are frequently used to convey information on the spatial extent of objects, i.e. the geometric shape of the object and its location. For example, if theaters are defined in the service class or in the context class, they may be linked say by a hasSpatialExtent inter-module role to the point or the polygon class in the space module to provide their geographical location and support proximity queries. Spatial objects (i.e. objects having a spatial extent) may be handled by a GIS, which can provide many computational services to the LBS to satisfy queries that call for such computations (e.g., a "nearest cash dispenser" query). Similarly, the temporal classes in C^{time} are the classes that convey the temporal features supported by the ontology software. Temporal features are usually defined to include instant, interval, etc. Associated to a class describing objects, for example using a hasTemporalExtent inter-module role, they denote the objects in the associated class as temporal objects, i.e. objects whose temporal extent is characterized by a temporal subset of the timeline underlying the ontology. Temporal feature are equipped with reasoning rules that support computational services such as computing the overlap between two time intervals. Examples of temporal objects include local calendars, local festivals, and temporal references such as Christmas, the "Jeûne fédéral" in Switzerland, or the coming Montreux Jazz festival. These temporal references can be used in queries, defining the timeframe of interest. The same object class can convey both a temporal and a spatial reference. For example, the "Montreux Jazz festival" denotes a place, Montreux, and a timeframe, in July. It supports query predicates such as *during/before Montreux Jazz festival*, and *near the main-site of Montreux Jazz festival.* Spatial and temporal modules are described in the sequel of this chapter.

The context module describes generic local knowledge that is relevant for improving the selection of appropriate services in response to users' queries. Examples of context knowledge include local weather and traffic conditions, events, and local specificities such national holidays. The context module is discussed in chapter 5.

3.2.2 Class Taxonomies

The skeleton of an ontology is its class hierarchy, expressing a taxonomy of concepts. In our LBS framework, the core ontology is a modular ontology. Each module is a self-standing ontology, equipped with the capability to hold and use inter-module links (roles) to enrich, whenever needed, its knowledge and reasoning capabilities beyond the actual boundaries of the module. Each module holds its own class taxonomy. When defining a class, the module it belongs to has to be specified. Whatever the module they belong to, classes are defined by one or more axioms. We build on the elementary axioms below to characterize a class as belonging to one of the following categories: atomic class, enumerated class, discriminated class, and set-based class.

DEFINITION 3.2. Module Class. *A module class c is a class in C (the set of all classes in the core ontology) explicitly defined as a triple (c, module, axioms) where c is the name of the class, module is the name of the ontological module the class belongs to, and axioms is the set of axioms that concur in defining the class. Axioms are either elementary axioms stated according to one of the following possibilities or complex axioms where class names are replaced by class definitions:*

- *c. This atomic axiom defines c as a new class without relating it to any of the other defined classes. If axioms for c only contains an atomic axiom, c is an atomic class and is direct subclass of* $\top$*, the top concept in the respective module, i.e.* $\neg\exists c_i,\ c_i \in C,\ c_i \neq \top,\ c \sqsubset c_i$.
- $c \equiv i_1, \ldots, i_n$. *This enumeration axiom defines c as an enumerated class where* $i_1, \ldots, i_n$ *are the set of individuals in the module stated by the axiom as belonging to the class.*
- $c \equiv c_j \sqcap$ *discriminator(c). This discriminating axiom defines c as a subclass of the class* c_j *belonging to the same module as c. discriminator(c) is a class definition that defines a membership criterion for individuals of* c_j *to belong to c. Typically, discriminator(c) is a restriction on the roles defined on* c_j.
- $c \sqsubseteq c_j$. *This inclusion axiom also defines c as a subclass of the class* c_j *belonging to the same module as c. This definition does not allow inferring which individuals in* c_j *belong to c.*
- $c \equiv c_i \sqcap c_j$, $c \equiv c_i \sqcup c_j$, *and* $c \equiv \neg c_i$. *These derivation axioms define c as a derived class whose individual are determined by the specified set operation:* $\sqcap(Intersection)$, $\sqcup(Union)$, *and* $\neg(Complement)$ *on classes* c_i *and* c_j *belonging to the same module as c.*

Atomic classes. We call atomic classes the classes that are direct subclasses of $\top$. Atomic classes play a fundamental role in the design of an ontology. They are the ones that precisely define the extent of the domain covered by the ontology, and serve as the semantic root nodes of the ontology (while $\top$ serves as the syntactic root). All other classes participate in structuring and refining the domain of discourse defined by the atomic classes. Atomic classes are classes defined by an atomic axiom (or classes appearing in an axiom and having no definition) and such that no other axiom in the ontology leads to infer they are subclass of another defined class. For example, let us assume the designer of the service module defines the Bus and Train classes using only an atomic axiom per class. In the Service module these two classes will appear as atomic classes, subclasses of $\top$. If some time later the designer creates a new class Transport using a derivation axiom $Transport \equiv Bus \sqcup Train$, Transport will be positioned in the Service module by the DL reasoner as a subclass of $\top$ and as a common superclass of Bus and Train. Consequently, Bus and Train will no longer be atomic classes.

In textual OWL specifications atomic axioms are stated using the owl:Class axiom specification. For instance, the following atomic axiom defines a class named "TransportService":

```
<owl:Class rdf:ID ="TransportService"/>
```

The axiom adds the class to the ontology. The class can be further characterized by additional axioms defining its relations, properties, restrictions and individuals.

Enumerated classes. Similar to their definition in databases, enumerated classes are classes whose instances are defined by enumerating them. These classes materialize the informal concept of repertoire. They are used to specify elements of the domain that are very specific to the context of the targeted applications. For example, an enumerated class can hold the names of the diploma delivered by a specific university, or the set street names in a given city. In a Swiss LBS service module, an *InternationalRail* class intended to describe major train services between cities in Switzerland and cities in neighbor countries may be defined as a subclass of the *Railways* class and be instantiated by an enumerated class definition, as follows:

$InternationalRail \sqsubseteq\ Railways$

$InternationalRail \equiv \{$"TGV", "DB", "Cisalpino", "ÖBB"$\}$

The same definition can be achieved in textual OWL using the oneOf property axiom to exhaustively list the desired instances. As shown in the example, a class defined by an enumeration axiom can also be defined by another axiom.

Discriminated class. We say a class c is a discriminated class if it is related to another class c_{sup} (its superclass) by an inclusion axiom ($c \sqsubseteq c_{sup}$)or by a discriminating axiom ($c \equiv c_{sup} \sqcap\ discriminator(c)$) where discriminator(c) is a class expression that constrains properties of the superclass c_{sup} to obey some given rules. Considering that ontologies are basically enriched taxonomies, most of their classes are discriminated classes. The explicit definition of a discriminator allows an automatic instantiation from the superclass to the subclass that complements the standard automatic instantiation from the subclass to the superclass. It is therefore preferable, whenever possible, to always use a discriminator in defining a subclass. Typical examples are easily found in the service module. For example, given an LBS Service module where rental services are available, the following definitions may hold:

Renting

RealEstateRental ≡ Renting ⊓ ∃hasRentalProduct.RealEstate

CarRental ≡ Renting ⊓ ∃hasRentalProduct.Car

ApartmentRental ≡ RealEstateRental ⊓ ∃hasRentalProduct.Apartment

The first axiom defines *Renting* as an atomic class, assuming *Renting* has no other defining axiom. *Renting* groups all rental services available in the LBS. The second axiom defines *RealEstateRental* as the subclass of *Renting* grouping all instances of *Renting* that are linked by at least one instance of the *hasRentalProduct* role to an instance of the *RealEstateRental* class. The third axiom similarly defines car rental services as another subclass of *Renting*. Finally, the fourth axiom defines apartment rental services as the subclass of real estate rental services such that the services have at least one offer for an apartment.

By exhaustively replacing the defined classes using their defining axioms, we may expand the axioms on a class to only include the atomic class and a set of discriminating restrictions. For instance, the *Apartment rental* axiom given above can be turned through concept expansion and subsumption into the equivalent axiom:
$ApartmentRental^{\dagger} \equiv \mathsf{Renting} \sqcap \exists \mathsf{hasRentalProduct.RealEstate} \sqcap \exists \mathsf{RealEstateType.Apartment}$

Assuming subclasses of *Renting* are defined as disjoint classes, given a query on ”apartment rentals” the LBS can prune the classes *HouseRental*, *ParkingPlaceRental*, and *CommercialPlaceRental*, so as to constrain the search space within the class *ApartmentRental*.

The discriminator can define a restriction based on cardinality constraints on a property. OWL provides built-in constraints on cardinalities, namely: owl:allValuesFrom($\forall$), owl:someValuesFrom($\exists$), owl:hasValue($\in$), owl:minCardinality, owl:maxCardinality, and owl:Cardinality. Using these constraints the above axiom defining the *Apartment rental* class is written:

```
<owl:Class rdf:ID="ApartmentRental">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#Apartment"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasRentalProduct"/>
          </owl:onProperty>
        </owl:Restriction>
        <owl:Class rdf:about="#RealEstateRental"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Set-based Class. We call *set-based classes* the classes defined as derived from existing classes on the basis of the set operations on classes. This allows deriving classes as union (owl:unionOf) or intersection (owl:IntersectionOf) of two other classes. In particular, an intersection between two classes describes a new class that is subclass of two different not disjoint classes, i.e. a class supporting multi-inheritance. A new class can also be defined as the complement of another class using the negation operator (owl:complementOf). In particular, negation can be used to define disjointedness, as already illustrated in previous examples. For example, the following axioms define a partition of transports into public and private transports

$\mathsf{PublicTransport} \sqsubseteq \mathsf{Transport}, \;\; \mathsf{PrivateTransport} \equiv \mathsf{Transport} \sqcap \neg \mathsf{PublicTransport}$

The derivation expression can generalize to any composite expression using set operators and class definitions as operands.

3.2.3 Relationships

Like for any structured repository, the semantics of ontology content relies both in the information nodes (here, the classes) and in the ways these nodes are linked (here, OWL object properties or DL roles). To analyze what kind of links we feel are needed in the LBS core ontology, we follow the conceptual modeling perspective, speaking of links as relationships. The rationale for the terminology change is not to be constrained by the limited kind of links that are currently supported by ontology models.

Is-a relationships form the backbone of any ontology and are a basic feature of all ontology models. We believe that in fact what is needed for correct knowledge management rather is *Discriminant Is-a relationships*, i.e. Is-a relationships whose definition includes the specification of the discriminating conditions that make an element in the superclass also an element in the subclass. The semantics of any Is-a relationships is the well-know population inclusion semantics, and its inferred property inheritance mechanism. A complementary backbone to build semantically rich data structures is the *part-of relationship* between a composite class and its component classes. In our approach, part-of relationships (equipped with cardinality constraints, as any kind of relationship) are synonym to whole-part, composition, and aggregation relationships discussed by other authors. The semantics of part-of relationships is that the composite class describes a whole that may alternatively be seen as a composition of parts of different types. Part-of is a binary relationship between the composite class and one of its component classes. This does not prevent the composite class to have components in different classes.

These two categories of relationships are discussed below. Other kinds of relationships will be introduced in the following chapters, as they are seen to be more relevant within a specific module, such as service, context, and user modules.

Is-a relationships and Discriminant Is-a relationships. Inherent to ontologies is the structure defined by inclusion axioms on which subsumption reasoning is based. In database terms, these axioms represent Is-a relationships between a subclass and a superclass. Because of its population inclusion semantics, properties and constraints attached to a superclass also hold for instances in its subclasses, while subclasses may be additionally characterized by their own properties and constraints. Is-a relationships in conceptual database schemas usually do not specify the membership predicates that characterize the different subclasses of a given superclass. In other terms, the criteria that distribute the superclass population among the subclass populations is not known. This lack of knowledge leads to unnecessarily long search process when looking for instances satisfying a predicate that corresponds to the membership predicate. As LBS care about quickly finding an answer to the user query, we favor for the core ontology an approach that allows to include the specification of membership predicate. We call discriminant property the property that discriminates instances of the superclass into populations of the subclasses. We call Discriminant Is-a relationships an Is-a relationships that explicitly bears a discriminating criterion. Thus, Is-a and Discriminant Is-a relationships are the basic links between classes in the core ontology.

DEFINITION 3.3. Is-a relationships. *An Is-a relationships is a partial order relation $\sqsubseteq$ on a set of classes such that $C_1 \sqsubseteq C_2$ implies that every instance of C_1 is an instance of C_2.*

The notation $C_1 \sqsubseteq C_2$ may be equivalently read as C_1 is a subclass of C_2, C_2 is a superclass of C_1, C_1 is contained in C_2, and C_2 contains C_1.

DEFINITION 3.4. Discriminant Is-a relationships. *A Discriminant Is-a relationship is a strict partial order relation $\subset_D$ on a set of classes such that $C_1 \subset_D C_2$ implies that $C_1 \equiv C_2 \sqcap D$, where D is a restriction on C_2 and $D \neq \top$.*

DEFINITION 3.5. Sibling Classes. *Two classes C_1 and C_2 are said to be sibling classes if both are related to the same class C_3 via Discriminant Is-a relationships $C_1 \subset_{D_1} C_3$ and $C_2 \subset_{D_2} C_3$ ($C_1 \equiv C_3 \sqcap D_1$, $C_2 \equiv C_3 \sqcap D_2$), and the restrictions D_1 and D_2 are defined using the same discriminating property but different values.*

In the simplest case, the restriction D is composed of a property and its value constraint. Let us consider again the apartment rental class defined by a Discriminant Is-a relationship with the RealEstate class that uses the RealEstate property RealEstateType as discriminating property:

$\mathsf{ApartmentRental} \sqsubseteq \mathsf{RealEstate} \sqcap \mathsf{RealEstateType.Apartment}$

A sibling to this class could be a cottage rental class defined as:

$\mathsf{CottageRental} \sqsubseteq \mathsf{RealEstate} \sqcap \mathsf{RealEstateType.Cottage}$

Conversely, the class

$\mathsf{HolydayRental} \sqsubseteq \mathsf{RealEstate} \sqcap \mathsf{RentalDurationUnit.Week}$

defining holiday rentals as real estate rentals whose rental duration is counted in weeks, is not a sibling of the ApartmentRental class because its definition uses a different property of the RealEstate class as discriminating property.

Whenever the discriminating property is functional (e.g. a real estate belongs to only one real estate type, or when using an allValuesFrom restriction with disjoint data ranges), it is inferable that the sibling classes based on this property are disjoint.

While Is-a and Discriminant Is-a relationships are directly expressed in DL and OWL axioms, finding the potential siblings of a given class needs reasoning on the definition of all classes defined as discriminated classes. It may be worthwhile, in order to expedite query processing, to introduce an explicit sibling relationship between two classes, implemented for example as a predefined role (object property) with the reserved name "Sibling".

Disjoint Relationship. Similarly and for the same efficiency reasons that lead us to introduce a Sibling relationship, we suggest introducing a Disjoint relationship between classes to directly state disjointedness between two classes. This can be implemented as a predefined role (object property) with the reserved name "Disjoint". At the conceptual level the Disjoint relationship can be defined as an n-ary relationship:

DEFINITION 3.6. Disjoint relationship. *A Disjoint relationship is a relation $\bowtie_D(S)$ where S is a set of classes, $S = (s_1, s_2, \ldots, s_i)$,$(i \geq 2)$, and for any two classes s_m, s_n in S, it always holds that $s_m \sqsubseteq \neg s_n$.*

This n-ary form is nothing but a shortcut to the specification of the corresponding set of binary disjunctions.

Query processing in the LBS can benefit from knowing that a set of classes are disjoint. For example, if the LBS finds an appropriate service in a class, it will not expect finding similarly appropriate services in classes that are disjoint to the first one. This avoids useless searches. For example, if an appropriate health care service is found in the class Hospital, the LBS will not further look into the class PetClinic, as PetClinic is defined to be disjoint with Hospital.

Part-of Relationship. Any description of real world data is likely to have to support descriptions of the same things at different levels of abstractions, corresponding to different perceptions and leading to different representations. A very typical example is the whole versus set of parts duality: in one perception, a real-world entity is seen as a whole and described as a single object; in another perception, the same entity is seen as a composition of parts that are individually identifiable and each one described as a separate object. In the LBS framework, we may have composite services, e.g. a HouseCleaning service that denotes a global service that actually consists in a grouping of individual underlying services, e.g. RoomCleaning + WindowCleaning + CarpetCleaning services. Similarly, a context class Environment may represent a global view of a set of context classes AthmosphericData, TrafficData, etc. where each of the latter classes covers a specific sub-domain in the environment domain. This corresponds to a perception where the environment domain is composed of several sub-domains.

We use the part-of term to denote the relationship that link the class representing the whole object with the class representing objects composing the whole. Classes may be organized into part-of hierarchies, such that one level in the hierarchy conveys a more synthetic perception and the lower level coveys a more detailed perception.

DEFINITION 3.7. Part-of relationship. *A Part-of relationship is a binary relationship between two classes C_1 and C_2, noted $\Subset (C_1, C_2)$, such that C_1 represents a whole and C_2 represents a part of this whole.*

These relationships are described in DL as roles between two classes (and in OWL-DL as object properties). These roles are normal roles, in the sense that they have no distinguishing characteristic. To keep the part-of semantics, we propose that these roles bear the reserved name hasComponent, and their inverse the reserved name isComponentOf. Cardinalities defined for these roles complete the definition of their semantics, specifying whether components may be shared by different composites, depend for their existence on the composite, are necessary or optional components for the composite.

The same composite object may be decomposed in different ways. For example, a century can be decomposed into years or into decades. This entails that it is possible to have many Part-of relationships between a composite class and sets of component classes. Therefore not to loose the semantics of aggregations we need a way to state which components belong to one given decomposition (i.e. cluster components by decomposition). To achieve this we introduce the concept of cluster of components:

DEFINITION 3.8. Cluster of Components. *A cluster of components is a set of classes S_c such that each class in S_c is related to the same class C by a Part-of relationship where C is the composite and S_c contains the classes needed to compose C according to a given composition process. We use the notation $\Subset (C, S_c)$, where C is a composite class, and S_c is a set of component classes, $S_c = C_1, \cdots, C_m$, $(m \geq 1)$ such that for each class $C_i \in S_c \Subset (C, C_i)$ holds.*

Example 3.1. Home Clean vs. Room Clean, Window Clean, and Carpet Clean.
$\Subset$(S, S_w)
Whole Service: HomeCleaning,
Component Services: RoomCleaning, WindowCleaning, and CarpetCleaning.
$\Subset$(HomeCleaning, {RoomCleaming, WindowCleaning, CarpetCleaning})

Other variants of the Part-of relationship will be introduced in the following chapters, due to their service-orientation and context-orientation.

3.2.4 Attributes

In conceptual modeling, each object type and each relationship type may be described by associated attributes and methods. Methods are not part of the ontological world, so we will not discuss methods in this work. Conversely, attributes are available, in DL as roles between a class and a data value domain and in OWL as data type properties. The main difference between the database and the ontology view of attributes/properties is that the former view an attribute as only existing as component of something (its "owner" construct) while the latter sees it as an independent thing, just like classes and roles. In databases an attribute is by definition a component of a single composite construct, be it an object type, a relationship type or another attribute. In an ontology it is possible to define a role/property independently of any specific relationship with other ontology items. Specifically, in OWL-DL a property can exist independent of the classes, and is defined as a tuple *([domain]*[4]*, property, [range])*. Syntactically, domain is a built-in component which, if it exists, links the property to one or more class descriptions (e.g., a union of classes). Similarly, range is a built-in component which links a property to either a class description or a data range. OWL-DL provides built-in axioms to specify the characteristics (symmetric, transitive, inverse, ...) and cardinality constraints that may hold for a property.

According to the characterization of attributes in [PSZ06], the attributes in databases can be: either *simple or complex*, either *mandatory or optional*, and either *monovalued or multivalued (of type set, list, or bag)*. Attributes in ontologies are simple, either mandatory or optional, and either monovalued or multivalued with an implicit set collection type. Complex attributes are not supported in ontologies, consistently with their binary relationship view of the world. Thus, whatever in a conceptual database view would be a complex attribute is defined in an ontology as a class.

In this subsection we briefly discuss attributes from the LBS perspective. We point out a specific characterization for service description attributes, which we call input attributes to convey the semantics that these attributes denote information that has to be provided when invoking a service to be able to operate the service. This allows the LBS query processor to determine a priori if a service found to be relevant can actually be invoked given the information available within the query. Finally, we introduce some additional attribute categories that play a specific role in the description of services in a LBS framework.

Cardinality Constraints. In a database context, given its closed world assumption, a *mandatory attribute* conveys a constraint on the creation of the object/relationship/complex attribute the attribute belongs to.

[4]Brackets [ ] specify the component as optional.

This cardinality constraint forces newly created instances/values of these objects/relationships/complex attributes to provide a value for the mandatory attribute. For example, if a Student object type holds a mandatory stcard# attribute, it is not possible to enter a new student instance into the database if the value of her/his student card number is not known. The same holds if a mandatory participation of an object type into a relationship type is specified: object instances can be created only if an instance of the relationship type is simultaneously created for this object instance. Ontological reasoning is based on the open world assumption. This makes mandatory constraints inoperable, despite the fact that it is possible to define minimal cardinality restriction. It is possible to create objects with no value for a mandatory property, simply because the reasoner assumes that the missing value may exist in the real world. Its temporary absence from the ontology is episodic and does not contradict the axiom defining the property. Actually, OWL-DL provides cardinality constraints restrictions (i.e. owl:minCardinality, owl:maxCardinality, owl:Cardinality) which allow defining the desired range for the number of values for a property of a class. Properties are by default multivalued, but the functional property axiom defines a property as monovalued, independently of any specific source domain. It is called a global cardinality constraint because it holds no matter which class the property is applied to. A *mincardinality* specified as 0 and a missing specification of a minimal cardinality denote that the associated property may not exist.

We believe LBS would benefit from the capability to specify and verify cardinality constraints. For example, LBS could use mandatory properties to define the necessary conditions for service individuals to be added into the available services. Similarly, properties that appear in the discriminator associated to the definition of a discriminated class typically are mandatory properties as knowing their value is needed to evaluate to which discriminated class an individual of the superclass belongs. Service descriptions could more precisely define their usability using mandatory properties, as explained below.

Input Property. We call *input property* is a *mandatory property* that expresses a constraint on the accessibility of the class it is attached to. More precisely, when querying the class, the class is considered to be not visible (hidden) if the query does not specify a value for each input property. Input properties of a class can be regarded as mandatory attributes of the query accessing the class. They are a valuable specification in particular if attached to a service class. In this case the semantics is clearly to define which attributes have to be valued in a query to make sense accessing the service. For example, a query to find a shop selling a product does not make sense if the type/name of the product is not specified in the query. Similarly, any query for a flight-ticket has to specify at least the departure and destination cities, possibly also the day/time of the desired flight.

Simple Attributes vs. Complex Attributes. In traditional conceptual modeling, attributes are qualified as simple (or atomic) if they directly hold a data value, if monovalued, or a collection of data values, if multivalued. Conversely, attributes are said to be complex if they do not directly hold a value but are used to denote the set of their component attributes that in turn are either simple or complex. Complex attributes are very useful when implementing conceptual modeling specifications to materialize attributes that have non-traditional, non-supported semantics. For example, to implement a time-varying simple attribute, e.g.

address, in a representation model that does not support time-varying attributes, the attribute is turned into a multivalued complex attribute, e.g. temporalAddress, with two components, one to define the actual value, e.g. address, and one to define the corresponding time, for example the day when the address started to hold. In our LBS we are specifically interested in supporting four types of complex attributes: composite property, multi-resolution property, dependent property, and range property. To represent these specific semantics in OWL-DL we need to add one component to the original complex attribute, say the semantic component, whose value domain is the enumerated set of supported semantics.

Composite property. As its name suggests, the composite property is composed of a set of properties such that none of the component properties can individually represent the semantics of the composite property. For instance, property *price* is composed of two simple properties *amount* and *currency.* Either amount or currency can not individually define the property *price.* Both are needed to specify the semantics of price. Either of them can be a simple property or a complex property.

Range property. A range property is a property whose value is a value range rather than a single value. It has two components: the minimal value and the maximal value of the property. For example, the *lunchTime* property holds a temporal range, e.g. 12:00, 14:30, with time granularity minute. Range properties are a kind of composite property, namely they are composed of two properties with the same value domain, and this domain is an ordered domain.

Dependent property. A dependent property is a property whose value depends on the value of another property or a fact. For instance, in car rental descriptions the *price* of the *carRental1* depends on the property *car-category* (i.e. all cars in the same category are rented for the same price), and the *deposit* of the *carRental2* depends on the fact whether the requester is a member of the EuropCar club or not (i.e. members of the EuropCar club do not need to provide a deposit while non-members need to provide a deposit). Dependencies are discussed in the next section.

Multi-resolution property. In spatial databases, multimedia databases and data-warehouses, for example, it is very common to represent the same object according to multiple resolutions. For instance, using different resolutions a given building can be characterized by a point extent or an area extent. Similarly, from the semantic viewpoint, an attribute can be represented at multiple levels of abstractions, i.e. different semantic resolution. For instance, the property *openingTime* can be a multi-resolution property. At a coarser level of abstraction it may only convey 'from Monday to Saturday'. At a finer level of abstraction, it may convey more details on the opening period for each working day, e.g. 9:00-19:00 (Monday-Friday) and 8:00-18:00 (Saturday). Conceptual multi-resolution attributes are typically implemented as multivalued complex attributes composed of an attribute to hold the value and a second attribute to specify the corresponding resolution.

3.2.5 Constraints

In traditional data management, constraints are as essential as objects, relationships and attributes to build a semantically rich definition of the data. A specific and important role for constraints is to rule data consistency by providing the specification of admissible instances and data values. Narrowing syntactically correct data to the subset of admissible instances and values relies on the closed world assumption: What is in the database is true and nothing else is true. Instead, ontological approaches rely on the open world assumption, stating that what is in the ontology is only a subset of the true facts that exist in the outside world. False is restricted to denote facts that contradict what is already known. This is why consistency concerns in ontologies basically check for satisfiability (i.e. non contradiction). Some real-world constraints can be expressed using axioms that lead to contradiction if data that does not conform to the constraint enters the ontology. A typical example is disjointedness constraints, such as man and woman representing distinct populations, which can be expressed by inclusion axioms linking man to the complement of woman and vice versa. However, restrictions based on using the existential quantifier and minimal cardinality constraints behave differently in ontologies and databases.

In terms of constraints, LBS requirements are different from those of databases and ontologies. Databases are tuned to manage very large amounts of data, with significant amounts of new data being acquired every day. Maintaining correctness and consistency of the data is the guarantee that applications can operate safely. Constraints play an important role in filtering the data that comes into the database and controlling that later updates preserve their consistency. LBS do not really have such concerns. They are more concerned with the quality of the metadata they acquire or elaborate. They contain relatively small amounts of factual data. Factual data is present, for example within the context module to identify local spatial and temporal references of interest (e.g. local landmarks and local holidays). The bulk of factual data within the LBS is the detailed description of available services, stored in the local sources, which are in charge of their acquisition, control and maintenance. In this setting, constraints exist (as for any data management application) but are not critical. The amount of data to be checked is small, and data acquisition is basically done once, when the LBS starts operation. Only if the local turnover of services is high it may become interesting to introduce constraints to check that new and updated service descriptions provide correct information. However, even in this perspective constraints would rarely be normative, given that there is no standard definition of what is admissible in a service description. This is not to say that LBS do not need constraints. The focus is, however, on accessibility constraints. For example, LBS may need to define that an alcohol-selling service is only accessible to users whose age is at least the minimal age defined in the local context information.

Regarding constraints in ontologies, their role is usually limited and not normative. To this extent, LBS resemble ontologies. Indeed, LBS main concern is collecting higher-level knowledge, i.e. concepts, links and properties that provide the semantic background to understand user queries and find appropriate answers. Constraints, as we have seen, are mainly restrictions used to characterize subclasses so that queries can be addressed to the most relevant class. However, LBS radically differ from ontologies in their goal. Ontological approaches target knowledge collection and enrichment in a cooperative way, where ideally every contributor is welcome to add new knowledge anytime without any central control. The ontology system takes care of checking satisfiability, and provides automatic placement of concepts thanks to its reasoning facilities (namely, subsumption reasoning). LBS target a much narrower scope. The knowledge they collect is limited in size and most frequently comes from authorized sources (e.g. public institutions) and domain experts. The number of players in charge of knowledge acquisition is small and control is centralized rather than distributed. Another likely difference with ontology systems is in query processing. LBS target is to quickly find some relevant answers rather than computing all possible answers including those that appear possible thanks to incomplete knowledge. To quickly find relevant answers, the closed world assumption seems more appropriate as it limits the scope of searches to the data that is there.

In traditional data management some kinds of constraints are directly supported by the data model and can be expressed using the constructs of the data model. Cardinality and uniqueness constraints are examples of these embedded constraints. Expressing other constraints relies on some integrity constraint description languages (basically some form of first order logic) flexible enough to describe whatever ad-hoc constraint is needed by the application. The relational approach has emphasized a specific kind of constraint: Functional dependencies (FDs). FD may be defined as follows: Given a relation R, a set of attributes X in R is said to functionally determine another attribute Y, also in R, (written $X \longrightarrow Y$) if whenever two tuples of R share the same value for X they also share the same value for Y (also stated: each X value is associated with precisely one Y value). Relational functional dependencies are defined to apply within a single relation and form the basis of the relational design paradigm. For LBS relying on a different data modeling paradigm, the definition of functional dependencies needs to be revisited to constrain the value of an attribute based on the values of other attributes in the same or in different interrelated classes, as long as there is a monovalued path from the target attribute to the source attributes of the dependency. As a first approximation, the extension of the definition is straightforward. Consider joining the set of classes containing the source and the target of the FD, i.e. the X and Y attributes (the join is performed using the roles linking these classes together). The result of the join can be established as a relational table. The FD holds if it holds as relational FD in this resulting table. However, a deeper investigation is needed to formally define FDs in non-relational models. This effort is the topic of a companion work within the laboratory [CP07]. Hereinafter we use the term dependency to denote this concept of non-relational functional dependency. As an illustration, let us consider the following example: a class City describes local cities, in a multilingual country. City holds an attribute spokenLanguage giving the language officially used within the city. City is linked by a functional role isInLR to the class LinguisticRegion (each city is related to only one linguistic region). LinguisticRegion has an attribute officialLanguage stating what is the official language within the region. The dependency

(City.isInLR.LinguisticRegion.officialLanguage) $\longrightarrow$ City.spokenLanguage

expresses that all cities related to linguistic regions having the same official language speak the same language.

More interesting for an LBS is the capability to express that the language spoken in a city is actually the same as the official language of the linguistic region the city belongs to. This corresponds to the MADS concept of derived attribute. Derived attributes are attributes whose value is computed instead of being input by users. The definition of the attribute specifies the computation formula. The simplest formula sets the derived value to be equal to the value of another attribute elsewhere. This would be the case if City.spokenLanguage is defined as derived attribute, according to the computation formula City.spokenLanguage = City.isInLR.LinguisticRegion.officialLanguage where both sides of the equality are monovalued paths starting from the same instance of City. In OWL-DL terms this derivation would be expressed as: The range of the property hasOfficialLanguage with domain City would be defined for each City instance as equal to the range of the property hasOfficialLanguage with domain LinguisticRegion in the LinguisticRegion instance that has the value of its name data property equal to the value of the hasLinguisticRegion property in the City instance. The example illustrates the simplest derivation case, consisting in copying the derived value from elsewhere. How far we can go with derivation depends on the language available for expressing the derivation formula. Usually, such languages are a kind of data manipulation language. In simplistic words, what is derivable is data that can be extracted using an SQL command. This does not fulfill all requirements. Derivation rules may be complex and require algorithms and computations, which can only be expressed using a full-fledged inference rule language. For example, an inference rule could compute the transport means a person is using, based on the observation that a given range of movement speed for a person moving along a given trajectory corresponds to a given transport means (e.g., a regular average speed below 3km/hour could denote the person is walking, while an average speed greater than 3km/hour could denote the person is using taxi, unless the trajectory shows stops at bus stop locations, in which case the person is assumed using a bus). Knowing the transport means may influence the contextualization of a query by the moving person, in particular queries involving distance and traveling time criteria. The transport means may determine a specific context. For example, if the LBS infers that the user is traveling by train (given speed of movement and a trajectory constrained by the railway network), the user request for online music-listening service will be processed considering the on-train context as the communication networks may provide services of different qualities according to the location (e.g. in a tunnel) and speed (e.g. high-speed) of the moving user. The capability to express inference rules can make a significant difference for LBS quality of service.

Another very interesting feature for an intelligent LBS is the capability to tie up plausible combinations of data. For example, looking at user profile data we may wish to state that a preference for 5 star hotels is not plausible if the information on revenue level shows the user as a low-income person. This is a kind of soft constraint, recommending rather than inhibiting. Similarly, it would be appropriate to state that restaurants with outdoors sitting (a terraces) could be given priority oven indoors-only restaurants only if the weather is nice. Soft constraints, i.e. recommendations and preferences, are expressed in our approach as links between data in the service, user, and context modules, and are discussed and illustrated in the following chapter.

3.2.6 Multi-representation Modeling

Semantically rich and flexible data management calls for the capability to support and describe multiple perceptions and multiple representations of the same real world phenomena. This is particularly important for enabling LBS to achieve their contextualization and personalization goals. The multiple representation features of the MADS data model have been extensively described in [PSZ06]. They provide generic solutions to express:

- How information is organized (in terms of data structure). For instance, *connectionSpeed* information may be modeled as a data type property, with range {good, normal, bad}. Alternatively, it can be described by a set of sub-properties, such as *hasUploadSpeed*, *hasDownloadSpeed*, etc.

- How information is encoded (dimension and unit). For instance, the *hasAntiVirusSoftware* property can correspond to a Boolean value, or can more precisely hold the name of the anti-virus software. Spatial and temporal data can be represented in diverse granularities.

- How the information in diverse representations is associated to form a consistent perception of the real world, tailored for the actual processing requirements. For instance, assuming the user is visiting a museum and wants information on art works on display, very precise knowledge of the user position, i.e. precise (x,y,z) coordinates, is necessary to identify the art work involved in the query. Instead, when the user finishes the visit and inquires about the bus-stops near the museum, the coarser location of the user (e.g. in the museum) is sufficient for evaluating the query.

We use the class `Representation` and a property `hasRepresentation` to denote the multiple representations feature of data. Further, it can have sub-properties, such as `hasScale` for `Location` class, `hasUnit` (e.g. day or hour) for temporal class. To describe that a property is multi-represented, the representation information needs to be encoded in the property by declaring it as a representation-dependent property. The way to define this is similar to the way to define space-dependent properties in the upcoming section.

As a consequence, when expressing relationships between data elements, it is necessary to add the representation information to the involved class or property, in order to make clear what representation of information will be involved in the relationship.

Finally, while we advocate that multi-representation features are essential for LBS, we do not elaborate further on the issue as we simply suggest using the result of previous work done in the laboratory [PSZ06]. These results provide the needed facilities and we do not see requirements that would be specific to LBS and not covered by the proposed approach.

3.2.7 Temporal Module

Temporal and spatial modeling is intrinsically part of the knowledge description in an LBS. Being able to characterize the temporal and spatial features of ontology classes is therefore essential, whether these classes describe concepts, services, contexts, or user preferences. Everything may be temporal and spatial, simply because everything exists somewhere sometime. It is up to the designer to define for which things spatio-temporal features must be described and for which other things they can be ignored. Examples of interesting temporal features abound. Shops have regular open time. Online merchandise can be purchased anytime but only be delivered during a given time interval. Public transports have specific temporal schedules for each stop on each route. A restaurant may become a disco after a given hour. Temporal specifications may have different granularity, i.e. use different temporal units: *hour*, *day*, *year*. In this section we show how definition of temporal extents and temporal values for LBS needs is supported.

In our data management approach, the temporal modeling dimension is orthogonal to the other modeling dimensions. The temporal characteristics of a class can be added, deleted or modified without harming the other features in the definition of the class. This supports the idea that a temporal module can be developed independently from the other modules, while its goal is to provide other modules with all the capabilities needed to describe and use temporal features. In our temporal module, support for temporal aspects concentrates on three objectives: 1) To define the temporal abstract data types needed to precisely specify the kind of temporal values (e.g. instant, interval) associated to a modeling construct (e.g. a class, a relationship, an attribute). The set of temporal abstract data types is defined as a hierarchy of classes. 2) To define some generic temporal properties that will be frequently used in LBS data descriptions, e.g. opening hours for services. 3) To define the temporal concepts and relations that help in the formulation and evaluation of user queries.

Temporal Data Types. The basic building brick in supporting time is temporal data types. They define the data value domains that temporal specifications may use. Elementary temporal data types are generally referred to as *Instant* and *Interval* data types, the former defining values that represent a single instant in time while the latter defines a time interval between a starting instant and an ending instant. Instant and Interval data types are generalized into a generic *SimpleTime* data supertype. Collection of respectively instants and intervals form the *InstantSet* and *IntervalSet* data types, sharing a common *ComplexTime* supertype. Finally, *SimpleTime* and *ComplexTime* are given a common supertype, *Time*, root of the temporal data types hierarchy. An OWL implementation of these simple hierarchy of data types has been reported in [Sot06], where two disjoint OWL classes `TSimple` and `TComplex` are defined as disjoint subtypes of the most generic `Time` class, root of the temporal hierarchy. The `TSimple` class is populated with instances of `Instant` and `Interval`. The `TComplex` class is populated with instances of `InstantSet` and `IntervalSet` classes.

These basic definitions of the temporal data types support further definitions in the temporal modeling dimension, such as the definition of temporal properties and other context classes, for example the *atmosphericConditions* or *trafficConditions* classes that obviously represent time-varying phenomena.

Several time granularities (e.g. day, month, year, week) need the specification of a calendar system to fully acquire their semantics. Frequently, the Gregorian calendar is used by default. However, an LBS that may be used worldwide must be able to support other calendars than the default one (e.g. the Chinese calendar or the Islamic calendar). Typically, the definition of the calendar in use is part of contextual specifications, for example as a functional property `calendarType` with an enumerated value domain. Using the OWL built-in axiom owl:sameAs, the same day in different calendars (e.g. Chinese Lunar Calendar and standard Gregorian Calendar) can be asserted as Same individuals in the temporal extent.

Temporal Properties. Temporal properties are properties whose value is from a temporal data type. In the core ontology of LBS, the temporal features of any class may be encoded as temporal properties, subtype of the generic temporal datatype property `hasTime` whose range is `Time`.

For example, LBS may define service classes to hold a mandatory `hasOpenTime` property, which describes the time intervals where the service is open/accessible. Its domain restriction is `Service` Class and range restriction is `Time`. The actual value of `hasOpenTime` for a given service is either of type SimpleTime or type ComplexTime.

More complex temporal properties may exist. For example, there may be temporal properties whose value depends on another property or fact. An example may be a property `hasDeliveryDuration` of a delivery service that has different delivery delays depending on destination (e.g. Europe, Asia, and America). In this case, one way to define these properties in OWL is to relate them to the same superclass `hasDeliveryDuration`:

```
<owl:DatatypeProperty rdf:ID="hasDeliveryDuration">
    <rdfs:domain rdf:resource="#Delivery"/>
    <rdfs:range rdf:resource="&xsd;duration"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasAmericaDelivery">
    <rdfs:subPropertyOf rdf:resource="#hasDeliveryDuration"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasAsiaDelivery">
    <rdfs:subPropertyOf rdf:resource="#hasDeliveryDuration"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasEuropeDelivery">
    <rdfs:subPropertyOf rdf:resource="#hasDeliveryDuration"/>
</owl:DatatypeProperty>
```

Temporal References and Temporal Relations. In daily communication, people often use some conventional/cultural temporal terms to refer to a specific timeframe. We refer to these terms as temporal references. For example, the terms now, today, tomorrow, and Christmas have universal and agreed meaning. Now denotes an instant, while today, tomorrow, and Christmas denote an instant or an interval depending on the granularity in use. These terms should be understood by the LBS and therefore be included in the core ontology. Many other terms have similar temporal semantics but their precise semantics depends on the local context. For example, New Year is a different day in China and in the Western world. National Day depends on the country. Morning has similar semantics everywhere, but may be defined using a different time interval (e.g. it may be 7-12am for working hours and 9-12am for a conference schedule). Context-dependent terms will appear in the context data (discussed in Chapter 5). Temporal references may obviously be used in queries to express a temporal constraint, e.g. *Before Christmas.*

Generally, these terms are described as axioms with `Time` values. For instance, *tomorrow* is encoded as `Instant` temporal data type with granularity `Day` format, such as yyyy:mm:dd. The term *afternoon* can be encoded as an `Interval` with granularity `Minute` such as $\text{afternoon} \equiv \forall \text{t}\ (\text{after(t, 13:00)} \sqcap \text{before(t, 18:00)})$.

Each *temporal reference* in the core ontology can be defined in OWL-DL using two properties, one to hold its identifier (usually a string value, e.g. Easter) and a second one related to a temporal data type (`Time` or one of its subclasses) which represents the temporal semantics of the time reference.

The LBS may use the classical temporal relations (e.g. before, after, during, around, from ...to..., and at), mainly for matching the availability of services against the temporal selection criterion in the user query. The semantics of these relations is well known. It is recalled in the following table, where t and T are either time instants or time intervals:

3.2.8 Spatial Module

Spatial modeling is very similar to the temporal modeling, both in motivation and in the way it is handled. LBS cannot exist without spatial information, and locating objects in space is quite the same issue as locating objects in time. The major difference between the two is the greater complexity of space, a two- or three-dimensional environment depending on application requirements, while time is a one-dimensional environment. Defining a time extent means identifying a set of points on a line (the timeline). Defining a space extent opens up to a variety of possible geometric forms and their combinations. As for the temporal module, spatial data management is supported by a dedicated spatial module in the core ontology. The module focuses on three aspects: 1) To define the spatial data types used in LBS, 2) To define the spatial properties in LBS, and 3) To describe the spatial terms and relations used in query formulation.

Spatial Data Types. Generally acknowledged definitions about spatial data types have been elaborated by the Open Geospatial Consortium (OGC), and have been included in GML, OGC's standard markup language for GIS data description. The work by [Sot06] has defined OWL descriptions for a similar hierarchy of spatial data types, the one of the MADS data model. The hierarchy has a generic `Geo` type as root, who two subtypes, `SimpleGeo` and `ComplexGeo`, respectively gathering all atomic geometries and all more complex geometries. Basic `SimpleGeo` subtypes are `Point`, `Line`, and `Area`, while basic `ComplexGeo` subtypes are `PointSet`, `LineSet`, and `AreaSet`. `ComplexGeo` in itself denotes complex geometries, whether homogeneous (e.g. sets of points) or heterogeneous (e.g. a mix of points and lines). The definition of spatial data types provides the base for further definitions in the spatial modeling dimension, i.e. spatial properties and spatial relationships. Spatial data types come with many computational facilities (e.g. computing a distance, a buffer) that are essential to spatial data management. Unfortunately, current DLs have very little, if any, support for spatial data. While implementing spatial data using DLs is feasible (see e.g. [Sot06]), implementing spatial computations in a DL environment would be neither easy nor efficient. The reasonable approach is to associate the LBS with a GIS (or a DBMS with spatial data support). Hence, the spatial data types in the LBS core ontology are likely to conform to the definitions in the associated Geo-software.

Spatial Class and Spatial Property. Alike for temporal properties, intrinsic spatial properties may be used to describe the spatial features of classes. We define a basic spatial property named `hasSpace`, serving as common superclass to all other spatial properties. In particular, thinking of services from the geographical perspective, each service has at least one spatial property, called `hasServiceSpace`, that denotes

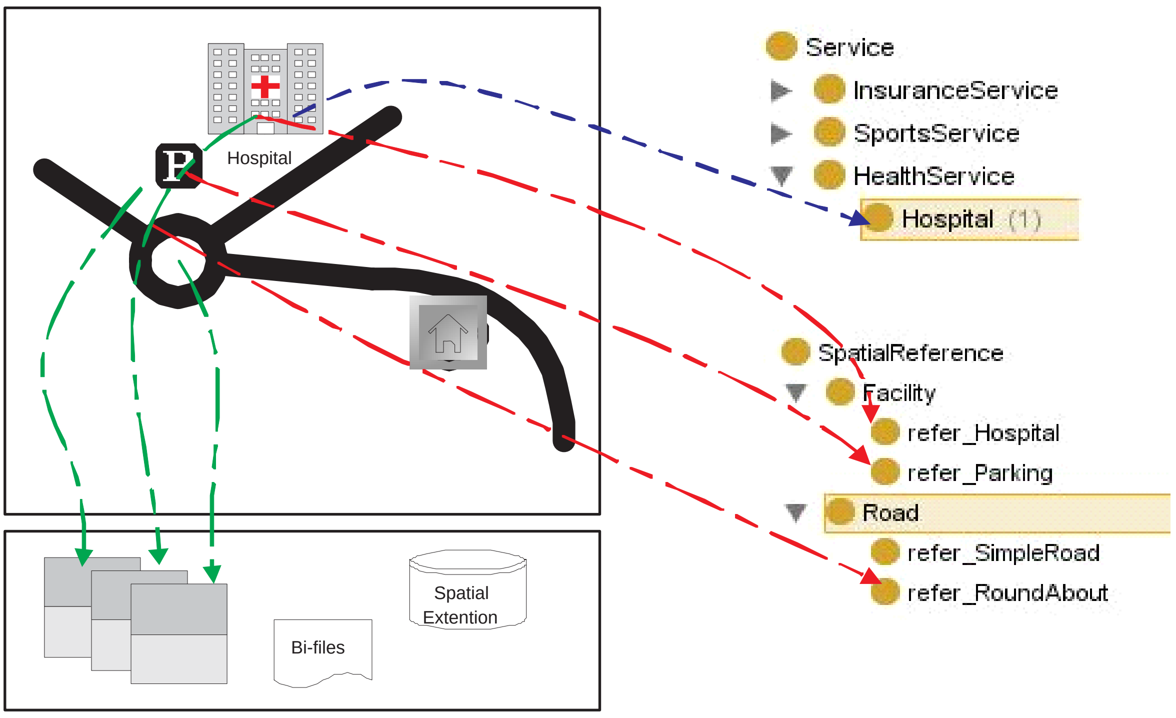
Table 3.1: LBS temporal relations and their logical expressions.

Temporal Relations	Logical expressions
before(t, T)	t < T
after(t, T)	t > T
at(T)	t = T
fromTo(t, T1, T2)	t≥T1 ∧ t≤T2 (T1<T2)
during(t, T)	t ⊂ T
around(t, T, △)	t≤(T+△) ∧ t≥(T-△)

the geographic coverage of the service. For instance, a restaurant can provide food service at its location which can be regarded as a `Point` spatial data type, a bus service can provide public transport service along a set of polylines within a given timeframe, a delivery service can disseminate commodities within a given region.

When encoding the spatial property value of the service into the associated Geo-software, the LBS keeps a self-defined value type (the pointer `id-space`) in the spatial facility as follows:

```
<owl:objectProperty rdf:ID="hasServiceSpace">
    <rdfs:subClassOf rdf:resource="#hasSpace"/>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range rdf:resource="#GeoReference"/>
</owl:objectProperty>
```



Legend:

Right-top: Services in LBS ontology;
Right-bottom: Spatial references in a spatial ontology;
Left-top: Spatial objects in Geo-map;
Left-bottom: Geo tools and engine, including shape-files, index, DBMS spatial extensions, and other GIS functions e.g. networking.

Blue-line: Mapping between the spatial property of a service in LBS and its correspondence in Geo-facility;
Red-line: Mapping between the spatial property of reference object and its correspondence in Geo-facility;
Green-line: Link between spatial object and its geometry file which is manipulated by Geo-facility.

Figure 3.2: Using spatial references to connect the core ontology and the external Geo-software.

Figure 3.2 illustrates this relationship and interactions between the Geo-software and the LBS ontology.

The `GeoReference` class in the core ontology is used to associate the ontology with the geometry data in the Geo-software. In the following example, both BuslineRefWeekend and BuslineRefWeekday are subclass of GeoReference.

The spatial property may depend on other properties or facts, as discussed for temporal property. Similarly, we apply the same strategy by adding a new class for describing the depended property or fact. For instance, the bus service range may change with time, e.g. at weekends it only runs on part of the whole route it travels on weekdays. In OWL, we describe this temporal-spatial property as follows:

```
<owl:objectProperty rdf:ID="hasWeekdaySpace">
    <rdfs:subClassOf rdf:resource="#hasServiceSpace"/>
    <rdfs:domain rdf:resource="#Bus"/>
    <rdfs:range rdf:resource="#BuslineRefWeekday"/>
</owl:objectproperty>
<owl:objectProperty rdf:ID="hasWeekendSpace">
    <rdfs:subClassOf rdf:resource="#hasServiceSpace"/>
    <rdfs:domain rdf:resource="#Bus"/>
    <rdfs:range rdf:resource="#BuslineRefWeekend"/>
    <owl:disjointWith rdf:resource="#hasWeekdaySpace"/>
</owl:objectProperty>
```

Spatial references and relations. The Geo-software associated to the LBS mainly provides three functions: 1) to keep and manipulate the geometry data of the service instances, 2) to identify and operate on reference places in query specification, 3) to handle the spatial queries, and provide more advanced services e.g. routing and map visualization.

Alike for temporal references, spatial references identify places users may refer to in their queries while assuming that these places are known to the LBS. For example, while in Lausanne a user query may ask for a hotel near the Olympic Museum. While at EPFL a user may ask if there is a cafeteria in building INJ. Olympic Museum and INJ building are spatial reference objects. As they have local semantics, these particular objects will be described as part of the spatial references in the corresponding context data.

Spatial references may be organized by their geographical distribution in the local grid, and by functionality or other criteria. This leads to consider that, in general, the `GeoReference` class contains both its identifier and geometry-type in the Geo-software, and its real world functionality such as museum, school, office etc. The precise geometry of the spatial coverage is represented in spatial data type as below.

```
<owl: Class rdf:ID="GeoReference">
<owl: dataProperty rdf:ID="hasIdentifier">
    <rdf:type rdf:resource="&owl; FunctionalProperty"/>
    <rdfs:domain rdf:resource="#GeoReference"/>
    <rdfs:range rdf:datatype="&xsd;string"/>
</owl:dataProperty>
<owl: objectProperty rdf:ID="hasFunctionType">
    <rdfs:domain rdf:resource="#GeoReference"/>
    <rdfs:range rdf:datatype="#FunctionType"/>
</owl:objectProperty>  ...
```

```
<owl: objectProperty rdf:ID="hasSpace">
    <rdfs:domain rdf:resource="#GeoReference"/>
    <rdfs:range rdf:datatype="#Geo"/>
</owl:objectProperty>
```

The spatial relations that the LBS shall support mainly rely on the associated Geo-software. In different spatial DBMS, spatial relations can be different. They usually include topological relations. Other types of spatial relations, e.g. directional relations, may be of interest to applications, although not necessarily supported by commercial Geo-software. Table 3.2 illustrates the most common spatial relations in LBS (i.e. near, within..., on, inside, along) and their correspondence in Oracle 10g.

Table 3.2: LBS spatial relations and their correspondence in Oracle.

Spatial Relations	Spatial operations in Oracle 10g
near(s, S)	SDO_NEAR
within(s, S, ℓ)	SDO_WITHIN_DISTANCE
fromTo(s, S1, S2)	SDO_ON
in(s, S)	SDO_INSIDE
along(s, S)	SDO_On
inDirectionOf(s, D, d)	No direct correspondence.

3.2.9 Shared terminology

The diversity of the culture, language and habits of LBS users and service providers, combined with the flexibility in query formulation and service description our LBS aims at (i.e. allowing users to express a query in their own natural language terms and similarly allowing service providers to describe their services in natural language terms) unavoidably leads to heterogeneity between the concepts and terms in LBS's core ontology and users' queries and data profiles. Hence, the LBS has to face the problem to understand and disambiguate the terms/phrases that show up in queries and service descriptions. The shared terminology repository is there to provide semantic assistance in this task. It extends the core ontology with additional concepts and definitions, playing the role of an enhanced dictionary. It provides synonyms and other relations between words/phrases, and identifies multiple interpretations of the same concepts/phrases.

To propose our formal definition of the shared terminology, we have first to distinguish between *term* and *contextualized-term* and explain the relations between the contextualized-terms.

- *Terms.* A term is either a word or a short phrase, as a headword[5] in the dictionary. The terms in the shared terminology are generally classified into two types: 1)**internal terms**. Internal terms refer to the terms directly imported from the core ontology, including the names of classes, the names of properties, and the names of instances (e.g. TGV is an instance of train service-class). 2)**external terms**. To enrich the vocabulary of the internal terms, we introduce the external terms from the external source(s) (e.g. an ontology or a thesauri, or a standard product catalog) or from the terms in user-queries after the LBS enters the operation phase.

[5]See the explanation in "Guide to the Use of the Dictionary" of The Concise OXFORD Dictionary, the 8th edition.

- *Contextualized-terms.* A contextualized-term is the association between a term and a context that uniquely and explicitly identifies a meaning for this term. Indeed, out of context, a single term often holds implicit and versatile meanings, which should separately correspond to diverse entries in the dictionary. A contextualized-term is composed of a single term and the context identifier, which in our approach denotes the source of the contextualized-term. For an internal term, its identifier is the core ontology (we assume the name of any concept in the core ontology is unique). For the external term, its identifier denotes the external source where the meaning is found. In the case the source has multiple meanings for the term, the identifier also denotes which unique meaning is the one associated to the term in this entry of the shared terminology. For example, contextualized-terms from WordNet[6] comprise a term and its synset.

- *Relationships between contextualized-terms.* Contextualized-terms may be linked by relationships that are useful to somehow extend the knowledge about a term in a given context. Typical relationships are semantic and linguistic relationships. For example, the synonym/equivalence relationship means that two contextualized terms share the same semantic meaning, so they can interchangeably be used, e.g. car-rental = hire-a-car, or car-rental = location-de-voiture.

We formally define the contextualized-term and shared terminology as follows:

DEFINITION 3.9. Contextualized-term in Shared Terminology. *A Contextualized-term in the shared terminology is a tuple t = (w,i), where w is a word or phrase, and i is the unique identifier of w's meaning, such that* $\forall\ t_m, t_n, and\ t_m = (w_m, i_m), t_n = (w_n, i_n), it\ holds\ t_m \neq t_n \longrightarrow\ i_m \neq i_n$.

Example 3.2. Some contextualized-terms in the shared terminology.

w	***i***	***definition in its source***
car rental	*♯CO ♯Car_rental*	*Rental ⊓ (∃ hasRentalProduct.Car)*
car rental	*♯WN ♯hire_Car♯(nounWordSense:1)*	*a rented car.*
celluloid	*♯CO ♯Celluloid*	*Medium ⊓ (∃ Usedfor.Photography)*
celluloid	*♯WN ♯celluloid ♯(nounWordSense:2)*	*a medium that disseminates moving pictures.*
...	...	
film	*♯CO ♯Film*	*Entertainment ⊓ ∃ldots*
film	*♯WN ♯film ♯(nounWordSense:1)*	*a form of entertainment that enacts a story...*
film	*♯WN ♯film ♯(nounWordSense:2)*	*a medium that disseminates moving pictures*
...	...	
film rental	*♯CO ♯Film_rental*	*Rental ⊓ (∃ hasRentalProduct.Film)*
...	...	
rental	*♯WN ♯rental ♯(nounWordSense:1)*	*property that is leased or rented out or let.*

The example above illustrates how we define the contextualized-terms in the shared terminology. Each row represents a single contextualized-term in the shared terminology. The first column labeled with w holds the word/phrase of the contextualized-term; the second column i holds the unique identifier of the contextualized-term; notice that the definitions of the contextualized-terms that appear in the third column are not kept in the shared terminology, but at their sources. To simplify the notation, *CO* stands for 'core ontology' and *WN* stands for 'WordNet'. The different components of the identifier are separated by '♯'.

[6]A lexical database for the English language. http://wordnet.princeton.edu/

The first part in the identifier stands for the source of the contextualized-term (e.g. core ontology, Wordnet, etc.), the second part is the label of contextualized-term in the source. Whenever a single term has multiple interpretations, it needs more information pinpointing the precise explanation of the contextualized-term in the source, e.g. the term's WordSense in WordNet. This is the (optional) third part of the identifier.

Using contextualized-terms provides many advantages: 1) they enable to provide (lexical, semantic or logical) interpretations on certain terms. 2) they enable to disambiguate the semantic differences between terms that are syntactically equal, cf. the many entries for 'film'. 3) they enable to enrich the vocabulary with lexical relations, e.g. synonyms 'celluloid' and 'film' from the WordNet. 4) they make it easier to analyze phrase patterns, e.g. 'xxx_rental'. Frequently, different terms from the ontology follow the same naming rules (i.e. the same pattern), e.g. 'DVD_rental' and 'car_rental' follow the 'xxx_rental' which denotes services consisting in offering some good for rental. Exploiting such patterns the ambiguity in a query may be reduced, in particular when the query cannot be matched exactly with the terms in the shared terminology.

In summary, contextualized-terms provide basic vocabulary support for the LBS query answering. But currently, the list-alike organization of the contextualized-terms is still insufficient to relate the contextualized-terms so as to facilitate the terms management and matching with queries. Formally, we define the shared terminology as a set of contextualized-terms and a partial function over pairs of contextualized-terms:

DEFINITION 3.10. Shared Terminology (ST). *The shared terminology is a tuple $<T,R>$ where T is a set of contextualized-terms, for each $t \in T$, $t = <w,i>$, and R is a set of relations over the contextualized-terms, $R \subseteq T \times T$ is a partial function from the set of all pairs of contextualized-terms* T *into a set of identifiers specifying whether the first term has a relation in* R *to the second term.*

Building the Shared Terminology. The process of constructing the shared terminology basically relies on the core ontology, i.e. the terms that define or characterize the location-based service. Then, with domain experts, the shared terminology can be further enriched by adding more terms and deriving the relations between them and with existing terms. Deciding which extra terms will be added and how they are related to the existing shared terminology, is the responsibility of the LBS designer and experts. Here we give an overview of the building process and then discuss three potential types of relations between contextualized-terms.

- **Step 1**: Import the class names and their identifiers from the core ontology to the shared terminology. We assume all classes in the core ontology are named in a literally meaningful way (e.g. 'restaurant'), instead of partial term or meaningless abbreviation (e.g. 'restau' or 'r1'). This helps to understand what function the service can provide.

- **Step 2**: Find out the naming patterns of class names. In many cases, the class names can not be simply described by a single word, but a short phrase, e.g. 'car rental'. We can expect that service providers follow some implicit naming rules when defining service classes. Similarly for LBS designers and the definition of context classes or user-related classes. These rules (or patterns) can be learned from some natural language processing techniques. Simply speaking, '*the patterns that we would apply center around a single word and incorporate a small number of words on either side*' [AR02]. For instance, 'car rental' adapts to the pattern 'product + business function'; 'car accessory' follows the pattern 'category + product' and so on.

- **Step 3**: Import the relevant terms from external thesauri and give their relationship with existing terms. The external thesauri can be a domain knowledge base, a lexical thesauri (e.g. WordNet), a data file added by the local experts, etc. Here we just discuss how to integrate the synonyms into the shared terminology. The first task is to find out the syntactic same terms from WordNet for terms of the shared terminology. Then we determine if they are semantic same or choose the semantic same one among several senses. Then, all synonyms and their identifiers are imported to the shared terminology and their relationships with the original terms are built up. Other terms and relationships can be integrated into the shared terminology in similar ways.

- **Step 4**: Refine the terms and relationships in the shared terminology. The refinement consists in cleaning the possibly repetitions in contextual-terms, authoring the definition for a term (when there are different definitions from various thesauri), and possibly allowing to modify, add or remove the contextualized-terms in the shared terminology as the strategy is really applied in LBS query processing.

Relationships in the Shared Terminology. As we discussed, the terms in the LBS core ontology are the starting point for building the shared terminology. The building process further enriches the terminology by incrementally involving the reachable external thesauri and ontologies. The new terms acquired from external sources are those terms found to be related to the internal terms by some relationships. These relationships belong to the following categories.

Linguistic Relationships. The main purpose of introducing linguistic relationships is to support semantic matching and reasoning rather than merely exact keyword matching. The most important terms imported into the shared terminology are the synonyms of the internal terms. External terms identified as synonyms are related with the corresponding internal terms by a 'synonym/equivalence' relationships.

Linguistic relationships also include relationships of a term with its hypernyms (terms with a more general meaning), hyponyms (terms with a more general meaning), mereonyms (terms describing parts of a term). What linguistic relations will be introduced in LBS considerably depends on the needs of the LBS. Currently, WordNet, a lexical thesaurus widely used in natural language processing and information retrieval, already has an RDF/OWL representation[7], which entails and standardizes more linguistic relations between words as important references for researchers and developers.

Contextualized Relationships. Due to the specific locality of the context knowledge used by an LBS, the terms in the current LBS can be different from similar and equivalent terms in another LBS covering a different region. Differences may relate to diversity of country, language (and dialect), culture, religion, etc. For example, *collège* in France means *secondary school*, while in Anglo-Saxons countries *college* means *university*. It is easy to get confused. Similar relationships can be explored by involving local contextual knowledge. For example, in Switzerland (only) the term 'Natel' is widely used as a synonym to 'mobile phone', yet *Natel* is a trademark of the Swiss telecommunication company *Swisscom*.

[7]http://www.w3.org/TR/wordnet-rdf/

Spatial/temporal Relationships. We agree that space and time are regarded as two significant types of context, as suggested by many classifications methods , e.g. [GS01] [HI04] [vBFA05]. But in our shared terminology, they are separated from the *Contextualized Relationships* because they function in a context-free mode for query matching in different dimensions, i.e. spatial and temporal. When specifying a spatial reference in a query, it is very common that the references are vague. For example, *St. François* in Lausanne can correspond to more than one reference. Let us assume there are three possible references such as *Church* St. François, *Place* St. François and *bus-stop* St. François. With the local knowledge, we quickly recognize that the *Church* and the *Place* can be easily taken as landmarks, and that for many queries the two are in fact equivalent landmarks, seen as denoting the same location (the church is actually within the square with the same name). However, it heavily depends on the subject of query and current location of the user. For instance, when one is asking for the bus information, the reference potentially means the *bus-stop*. Similarly, terms can be related according to their temporal relationships. For instance, the term *Easter Holiday* can be differently understood in different cultures, religions or countries. With the official calendar in Switzerland, the term denotes the period in days from *Good Friday* until *Easter Monday*. Hence, the temporal relationship between the three terms above can be defined as $Equal^T$(Easter Holiday, Between(Good Friday, Easter Monday)).

The organization of the shared terminology is sketched in Figure 3.3, which shows the types of relationships it contains: Linguistic relations, Contextualized relations, and Spatial/Temporal relations.

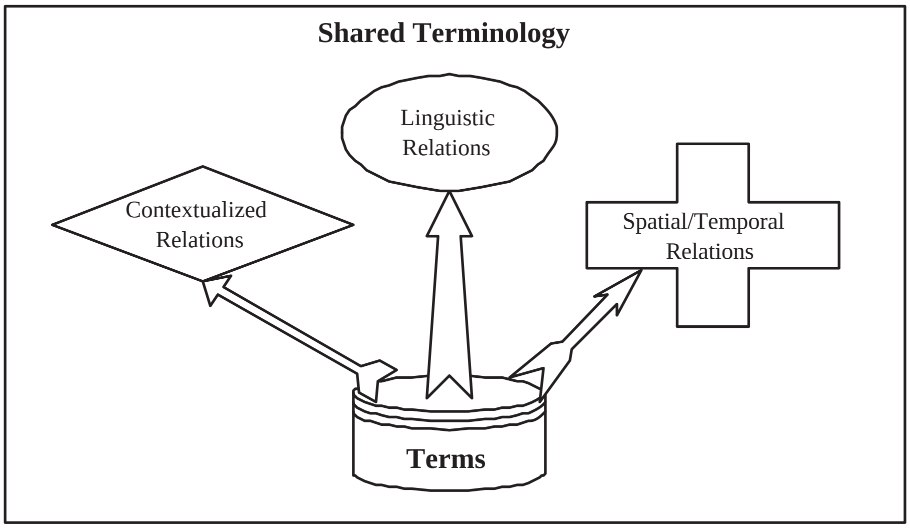


Figure 3.3: Composition of a Shared Terminology in LBS Data Infrastructure.

3.3 Evolution and Update of Ontologies in LBS

Evolution management is a serious challenge for any information system. Different from the database evolution, *when ontology evolves, we must consider not only the effect of ontology changes on the way applications access instance data, but also the effect of these changes on queries for the ontology contents itself* [NK04]. According to it, ontology evolution can be characterized as either traced or untraced. *"Traced evolution* *largely parallels schema-evolution where we treat the evolution as a series of changes in the ontology. After each operation that changes the ontology (e.g., add or delete a class, attach a slot to a class, change restrictions on slots, etc.), we consider the effects on the instance data and related ontologies, depending on the dimension of compatibility we use. The resulting effect is determined by the combination of change operations. With the untraced evolution, all we have are two versions of an ontology and no knowledge of the steps that led from one version to another. We will need to find the differences between the two versions in an automated or semiautomated way."* In particular, in our LBS setting, when either the core ontology or another knowledge repository evolve, we must consider the effect of these changes on queries for contents of the old core ontology, as well as the effect of these changes on the contents of the old mappings between the core ontology and other repositories. In our work, we suggest to deal with ontology evolution in the traced mode because: 1) in the untraced mode the core ontology will produce too much redundant data as the ontology evolves, which will result in less efficient query answering and knowledge reasoning. 2) in our infrastructure, any evolution basically involves two ontologies. In the traced mode, it is reasonable to address the issues of evolving interdependent ontologies; in contrast, in large distributed ontologies environment, complicated cross-dependencies between ontologies are a great challenge for the evolution strategy.

The work in [NK04] provides a sound background to handle the possible operations and their effects and corresponding manipulations on the ontologies in the LBS. On this basis, we investigate and extend certain potential operations as applied to the evolution of the core ontology, as shown in the Table 3.3. We identify the most common operations on the core ontology and the effects and appropriate manipulations to keep the ontologies consistent. Often, the execution of an operation potentially triggers one or several relevant operations, similarly to what happens in databases. In the literature, the study of ontology evolution has drawn much attention and some tools for ontology merging or mapping are available (e.g. Prompt, OntoMorph and ONION). We do not investigate the details of designing and optimizing the algorithms for each operation in the evolution and update of ontologies. This is beyond the scope of the thesis. We only illustrate some strategic solutions when some operations occur as shown in the following example.

Table 3.3 briefly summarizes the possible operations when the ontology(s) evolves and updates. The operations 1-16 show that the evolution of the metadata of core ontology potentially results in corresponding updates of the mapping ontology, e.g. to delete a service class A in the core ontology may result in modifying all mappings with the target element "service class A" in the mapping ontology. The operations 17-20 describe the updates of the mapping ontology when a service instance is added, deleted or updated. We will continue to discuss the maintenance of service profiles in LBS in the next chapter.

3.4 Chapter Summary

In this chapter, we first discussed the features characterizing data management in an LBS, i.e. locality, mobility, dynamics, heterogeneity, on the fly interoperability, and modularity. These characteristics call for a dedicated data management strategy, based on global decentralization (using peer to peer strategies) and a locally centralized approach to structure the service-related data. The rest of the chapter focused on

Table 3.3: Evolution and Update of Ontologies in LBS.

	Operations	Effects/Manipulations on $\mathcal{CO}$	Effects/Manipulations on $\mathcal{ML}$	Examples
1	Add a Service Class C_s	non	non	Add *CarRental* to $\mathcal{CO}$
2	Delete a Service Class C_s	put instances of C_s to C_s super-class	the targetElement values for all instances involving C_s are lost	delete C_s *CarRental*
3	Add a Property p	non	non	Add p deposit to $\mathcal{CO}$
4	Delete a Property p	the values of p for all instances are lost	the targetElement values for all instances involving p are lost	
5	Attach p to C_s	non	non	Attach p *'deposit'* to C_s *'CarRental'*
6	Remove p from C_s	the values of p for instances of C_s are lost	the targetElement values of all instances involving p of C_s are lost	Remove p *'deposit'* from C_s *'CarRental'*
7	Add a *subtype* relation between two service classes, such as $C_{sub} \subseteq C$	C_{sub} Inherits all properties from C	Mappings which apply to C can also apply to C_{sub}	Add the axiom CarRental $\subseteq$ Renting
8	Remove the *subtype* relation between C_{sub} and C	C_{sub} will loss all properties inherited from C, and loss all values of those properties.	Mappings which apply to C can no longer apply to C_{sub}, and the targetElement values of all instances involving properties inherited from C in C_{sub} are lost.	Delete the axiom CarRental $\subseteq$ Renting
9	Move a property p from a subclass C_{sub} to a superclass C	Class C and its all subclasses such as $(C, C_1, C_2, \ldots)$ will have property p	Mappings involving p in C_{sub} can also apply to p for $(C, C_1, C_2, \ldots)$	Move property *deposit* of class *CarRental* to *Renting*
10	Move a property p from a superclass C to a subclass C_{sub}	Class C and its all subclasses except C_{sub} will loss property p and all values of p in instances.	Mappings that apply to p of Class C and its all subclasses are no longer applicable except C_{sub}. And all targetElement values for all instances involving p of Class C and its all subclasses except C_{sub} are lost.	Move *deposit* from *Renting* to *CarRental*
11	Modify the restriction for a property p	Some instance values of p may be invalid with the new constraints.	Mapping instances can include the invalid values of p due to the new constraints.	*deposit*'s range changes from boolean $\longrightarrow$ currency.
12	Encapsulate a set of properties P attached to a class C_s into a new class C_{new}	Create the property relating C_s and C_{new}, copy values of all P in C_s to C_{new} and remove P from C_s.	In the mapping instances, the targetElement's range involving the encapsulated properties will change, but the targetElement's range involving the values of the encapsulated properties will not change.	Encapsulate properties *email*, *telephone* and *fax* in *CarRental* into new class *Contact*
13	Modify a simple property p to a complex property p_c in a class C_s (e.g. a multiple-representation property, dependent property, or composite property discussed in Section 3.2.4)	Modify the values in the complex properties.	Modify the targetElement in the mappings from p to p_c(or p_c 's property), and update all instances of p in the mapping instances with all corresponding values of p_c.	Modify property *car-type* in class *CarRental* from a simple property to a multiple-representation property, with two semantic representations 'Fuel-based' and 'Function-based'.
14	Change a complex property p_c to simple property(s) p in C_s	Attach property p to C_s and add the data values of p, remove p_c from C_s.	Mappings involving p_c and its values are lost, new mappings involving emphp and its values can be added.	Multiple-resolution property *deposit* changes to the currency type of values.
15	Merge service classes $C_1 + C_2 \Rightarrow$ C (In our work, we mainly focus on merging the sibling classes or merge class and its sub/super-class.)	Merge the class, properties and instances in classes $C_1 + C_2$, it may requires to modify some properties, their constraints and values, which will trigger some operations above.	Merges the mappings involve in $C_1 + C_2$, their properties and instances by refereing to the merging result of $C_1 + C_2$ in $\mathcal{CO}$.	*HouseRental* + *ApartmentRental* $\Rightarrow$ *HomeRental*.
16	Decompose a service class C $\Rightarrow C_1 + C_2$ (It means to generate two specific subclasses to replace old class C)	Attach the appropriate properties separately to C_1 and C_2,	Update mapping instances accordingly, targetElement change from C to (C_1 and/or C_2)	*HomeRental* $\Rightarrow$ *HouseRental* + *ApartmentRental*
17	Add a service instance $\mathcal{S}$ and its data $\mathcal{D}_s$	Add $\mathcal{S}$ to a class C_s in $\mathcal{CO}$, add $\mathcal{D}_s$ to properties in C_S, it may requires to trigger some operations above, e.g. add a property, modify a property, modify the constraint of a property etc.	Add the mappings of $\mathcal{S}$ and $\mathcal{D}_s$ to $\mathcal{ML}$.	Add *Car_Rental1* and its data to *CarRental* ...
18	Delete a service instance $\mathcal{S}$ and its data	Delete $\mathcal{S}$ corresponding service instance s in C_s, and may delete its properties' data	Delete the mappings between $\mathcal{S}$ and s, mappings of properties and values between them.	Delete *Car_Rental1* and its data to *CarRental* ...
19	Update the property's constraints and values of a service instance	Update the property instance of C_s in $\mathcal{CO}$	Update the targetElement value of property p in $\mathcal{M}_i$	Update the property *price*'s value in service *Car_Rental1*.
20	Update the property values of a service instance	Update the properties of instance in	Update the targetElement value of property p in $\mathcal{M}_i$	Update the property *price*'s value in service *Car_Rental1*.

the definition of the semantic data infrastructure we propose to take into account LBS requirements. We emphasized the split in data organization between knowledge that is elaborated by the LBS itself (the core ontology and the shared terminology) and knowledge that is provided by external partners (users and service providers). In parallel, we stressed the benefit of using a modular approach for the development of the core ontology, and identified the basic modules that are essential to LBS: service module, user module, context module, and space and time modules. The service and user modules hold abstractions derived from the service descriptions (created by service providers) and from the user profiles. They are meant to represent the essence of the available services and the interacting users. Mappings link these essential representations to the corresponding data in user profiles and service descriptions. The context module, elaborated in cooperation with the potential context sources (e.g. local administrations), holds the contextual data that may have an influence on the selection of services in response to user requests. Finally, the space and time modules provide the supporting concepts for description of spatial and temporal features in all the other modules and repositories within the LBS.

The core ontology represents the domain-specific conceptual views in a unified fashion. In addition, the shared terminology provides terminological support to overcome the heterogeneity problem that often results from the diversity of the cultures and languages, as well as from the lack of the knowledge of the LBS' structure and naming for users.

The goal of presenting a semantic infrastructure is to clearly identify the stake-holders of LBS data management. Clarifying the issues is the focus of this thesis. When adopting an implementation perspective, repositories may be organized differently in view of achieving best performance. For example, the core ontology and the shared terminology may be implemented as a single repository holding the syntactic and the semantic knowledge about the domain of the LBS. As another example, the space and time modules may be embedded in the capabilities of the system, rather than as components of the ontology. It will be up to the implementation team to revisit the semantic infrastructure for efficient implementation.

In the following chapters, we will delve into more details on the management of context and user profiles, on service-profiles management, and on the relationships between the modules in the core ontology and between the core ontology and the service and user profiles.

Chapter 4

Services and Data Profiles

4.1 Introduction

4.1.1 From Web Services to Semantic Web Services

Web Service technologies provide a new paradigm to share and communicate between heterogeneous and dynamic web applications. Each web service can publish its service functionalities, define the exposed interface (e.g. WSDL), and specify communication protocols (e.g. SOAP) to a service registry (e.g. UDDI). By looking up the registry or issue a query, a web service requester can find and further invoke/bind with the corresponding web service. Among the existing web services, Amazon is a typical example, where each service provider can publish their service/product information to Amazon website (i.e. the registry) according to its products' category, then the service/product requester can search and proceed the payment transaction with the chosen service provider. Additionally, a web service can seek and orchestrate with another web service to achieve a more complex web service, called *composite web service*. The communication means of web services requesters and providers, i.e. ”*publish-find-bind*”, enables web services to be borderlessly connected and communicate within the large community of web services. However, due to the lack of semantic consistency, web services still need the interference and interactions of human to some extent, to disambiguate the service descriptions, and to execute and compose web services in an appropriate manner. Consequently, the emerging *Semantic Web Services* (SWS) enhance the capabilities of self-explanation, automated discovery and invocation, composition or orchestration of web services.

To achieve semantic web services, it is necessary to define an appropriate conceptual model, well-operated infrastructure, a formal language with strict syntax and semantic specifications, potentially with definitions in communication and executions. In W3C community, different SWS infrastructures and descriptions have been proposed, e.g. OWL-S, WSMO, and SWSF.

OWL-S[1]. It is an OWL-based web service ontology as its name denotes. It consists of three parts of upper ontologies, i.e. service profile, process model, and service grounding, to describe different aspects of web services, i.e. *”what does the service provide for prospective clients?”*, *”how is the service used?”*, and *”how*

[1] Refer to the document http://www.w3.org/Submission/OWL-S/

does one interact with the service?". Its service profile ontology describes the information of service provider, service's functional features, input/output constraints, and preconditions/effects. In addition, an OWL-S service profile may be associated with property serviceCategory, which suggests a generalization/specialization relationship between services. The ServiceModel profile provides the details how a service can be interacted and executed, i.e. viewed as a *process*. Hence, the process profile involves a set of interaction parameters such as input/output, precondition/result, and participants. In addition, the service process model distinguishes the service processes in three types, i.e. atomic processes, simple processes, and composite processes. Furthermore, to facilitate the services composition and execution, process profile provides a set of constructs, e.g. sequence, split, split-join, any-order, if-then-else etc. The grounding profile describes how a service can be accessed, such as the protocols and message formats. OWL-S is based on OWL, a standard web markup language. The OWL language provides three increasingly expressive sub-languages for different uses, i.e. OWL-Lite, OWL-DL and OWL-Full. They are based on different logics, providing diverse expressiveness and having to deal with diverse complexities in decidable computations.

In OWL-S approach, each service is an instance of web service ontology. Unlike web services, OWL-S infrastructure does not explicitly define a registry model which a service can be advertised to. Instead, the service profile's declarative representation enable the services to be employed in different types of registries. For instance, with the serviceCategory property, a car-rental service can be associated with car service category of certain registries. However, how to associate a service to a registry is still an open question and may have to resort to human interference or the assistance of external ontology. Alternatively, there is no registry in p2p environment, where the decentralized SWS infrastructure calls upon different strategies of service matching and discovery.

WSMO/WSML/WSMX[2]. As another important initiative in the SWS community, WSMO (Web Services Modeling Ontology) is motivated at a different conceptual model to represent and manage semantic web services from OWL-S. Rather than defining the metadata of web services separately from function, process and protocol aspects, WSMO differentiates its top-level elements according to their roles in interaction and interoperation, i.e. ontologies, goals, web services, and mediators. *Ontologies* provide the formal semantic support for concepts used in all WSMO components; *web services* ontology defines the common elements in describing services, such as nonFunctionalProperties, usesMediator, and hasInterface etc.; correspondingly, *goals* ontology specifies the user's request in terms of nonFunctionalProperties, usesMediator, requestsCapability and requestsInterface etc.; *mediators* ontology is the core of mediating the heterogeneous elements between different WSMO components, ranging from the terminology mismatch, to protocol or process conflicts. Therefore, each top-level component in WSMO is independent of other components and it can communicate with other components with the assistance of appropriate mediators. In WSMO, goals ontology and web services ontology clearly express the service requests from clients' viewpoint and service provision from services' viewpoint. Furthermore, WSMO extends the goal ontology with service-quality specification,

[2] Refer to http://www.w3.org/Submission/WSMO/

enabling users to customize their requests, on the basis of the existing goals ontology [WSMO Final Deliverable D4.17]. In accordance with WSMO architecture, WSML is mainly based on F-Logic and provides the language support to describe the ontologies, goals, web services and mediators. In addition, WSML develops a set of language variants to investigate the applicability of different language formalisms. WSMX is specially designed for testing the practicability of the WSMO for SWS discovery, matching, orchestration, semantic interoperation and sources management. In comparison to OWL-S, WSMO/WSML/WSMX concentrate on investigating diverse means to providing a conceptual model, defining an appropriate formal language for the SWS, tackling the heterogeneous issues between ontologies with mediators, and examining their viability in a dynamic SWS execution environment. However, to define or find the right mediator for a new data source still demands the efforts in automated ontological alignment and matching.

SWSF[3]/SWSO[4]/SWSL. Influenced and inspired by OWL-S, SWSF is a more recent proposal of Semantic Web Services Initiatives. It mainly consists of two components: an ontology and its corresponding conceptual model, i.e. SWSO; a language used to specify the concepts and behaviors of semantic web services, i.e. SWSL. SWSO presents a conceptual model to better describe semantic web services in terms of ontology and presents a description of first-order logic axiomatization of the ontology, called **FLOWS** (First-Order Logic Ontology for Web Services). Similar to service profile in OWL-S, FLOWS includes the properties such as service name, service author, service URL etc. in *Service Descriptor*. Moreover, FLOWS enhances the descriptions of services process by extending its infrastructure on the basis of PSL (Process Specification Language). Consequently, FLOWS Process Model is composed of six ontology modules: FLOWS-Core, Control Constraints, Ordering Constraints, Occurrence Constraints, State Constraints, and Exception Constraints. Accordingly, SWSL is a language to formally describe web services concepts and individual web services. It comes up with two variants, i.e. SWSL-FOL and SWSL-Rules. The former is based on first-order logic and is used to specify the service ontology (SWSO); the latter is a logic programming language, which provides the language support for service discovery, communication, policy definitions etc. In addition, SWSL provides a platform to bridge the semantic transformation between SWSL-FOL and SWSL-Rules due to their common language base First-order Logic.

In addition, there are alternative approaches to specifying and inter-operating semantic web services, such as IRS-III[5] and WSDL-S[6]. Initiatives in SWS community concentrate on enhancing the semantics of web service descriptions in two key aspects: to define the conceptual models and infrastructures for automated SWS discovery, execution and choreography; and to specify the formal languages to define concepts, process and communication of SWS. Consequently, these approaches provide generic guidelines on SWS architectures and unified definitions for diverse applications' needs. OWL-S aims at defining a set of generic classes and properties, which can be easily referred in describing individual services and thus being inter-operated by end-users and software agents. WSMO goes further in modeling services into diverse domains/applications,

[3] SWSF: Semantic Web Services Framework, http://www.w3.org/Submission/SWSF/
[4] SWSO: Semantic Web Services Ontology, http://www.w3.org/Submission/SWSF-SWSO/
[5] IRS: Internet Reasoning Services. For the details, see http://kmi.open.ac.uk/projects/irs/
[6] Web Service Semantics. http://www.w3.org/Submission/WSDL-S/

separates the goal and service ontologies from user and service viewpoints, and takes into account the mediation issue in the architecture design. SWSL provides the more solid language support in service process specification than its ancestor OWL-S by smoothly integrating the logic programming into its infrastructure.

Due to the diversity of motivations of SWS approaches, their architectures, strategies of service management and requests processing are distinguished from each other. OWL-S does not present an infrastructure to manage SWS, instead, it specifies the individual SWS with a root concept *Service* and extends it with three ontologies. In other words, each individual SWS can be regarded as an automated and self-explained agent. Relying on the properties serviceCategory, and corresponding values in the service profile, individual SWS can be identified and related to each other, i.e. subclass/superclass. Similarly, according to the specifications in the *Service Process* and *Service Grounding* ontologies, OWL-S services can achieve the automated invocation, compositions and interoperation. In contrast, WSMO targets at exploring different ways to manage SWS, by designing a more full-fledged architecture to deal with different parts of SWS interactions and their interactions in SWS in various ontologies. In addition, WSMF explicitly defines registry-like components to publish and manage these ontologies.

The current efforts in SWS community are mainly concentrated on unifying the definition of SWS and designing a proper SWS infrastructure. In the next step, the researchers and practitioners will take a close look on SWS and potentially concentrate on some pragmatic issues, such as how to employ these infrastructures and definitions in different applications for diverse purposes, how to make efficient request processing and matching, how to standardize APIs to release application developers from reprogramming etc.

In particular, for LBS services management, we are concerned about not merely how to identify and maintain the individual services, but also how to propose efficient services management strategy in order to rapidly respond to user's requests. Therefore, we must take into account following issues:

- How LBS identify the heterogenous semantics of the service profiles and make them tangible to end users,
- How LBS organize the services data and make them inter-operable, such as a well-organized taxonomy (specialization/generalization) and commonly-used relations (e.g. disjoint, part/whole or orchestration),
- How LBS define the service profiles at the syntax level, e.g. mandatory properties like *spatio-temporal* characteristics of services. Moreover, definitions in our LBS are preferred to be compatible with the standard definitions of SWS community.

4.1.2 Our Approach to Describing and Managing Services

As stated in chapter 3, our LBS ontology is an ontology of services and service usability, rather than an ontology of abstract concepts. This is motivated by the feature "Locality, Mobility and Dynamics" of LBS. In other words, core ontology only contains service concepts that are available and relevant to current LBS, rather than generic ones specified in OWL-S. Our definition of service class is based on OWL-S service profile, i.e. each class has a set of properties. However, we extend the service definition in OWL-S with a set of predefined relations, e.g. *functionality similarity* and *orchestration*. These relations defined in our work are concerned about different aspects. For instance, *functionality similarity* relation describes different services having similar functionalities in certain context, which can help to offer satisfactory services to end-users even in case the exact matching can not be obtained. The services involved in a *functionality similarity* relation need to hold similar functionalities with certain constraints, but they do not necessarily have inter-operation. In contrast, Orchestration engages the simple or complex processes between services or inside a complex service. The orchestration relation is basically defined on the basis of some "state" information, such as pre-condition and post-effect etc. In both OWL-S, SWSO and WSMO present the definitions on the process of services using various ways and can be possibly integrated in our LBS. Our relation definitions are not contradictory to the definitions of service process in SWS proposals; instead, they are the important complement because they are some specialization of service processes defined in SWS proposals. Our definitions provide the guideline and facilities to LBS designers and developers, since they more explicitly specify the relation semantics than those generic ones in SWS definitions. Moreover, considering the influence of context/users on the service selection, we will further define the inter-module links and mappings between concepts in different modules in chapter 6.

Regarding the syntax of service profile in LBS, they mainly follow the definitions in OWL-S, and extend with certain mandatory properties, i.e. spatio-temporal semantics, e.g. OpenHour and ServiceCoverage. From the practical viewpoint, we introduce the concept *data profile*, which describes a data source consisting of a set of service profiles. A data profile can be the schema of a database or a website. In addition, we do not emphasize the automated inter-operations between the services, i.e. descriptions on service process and grounding, because we assume LBS mainly serve as an information provision portal and just have limited rights to execute/interact real services with representative of end users.

In addition, the service module of core ontology is seen as a service registry and its concepts are organized in a taxonomy. Different from the tacitly assumed taxonomy in OWL-S, our taxonomy is clearly specified in the core ontology. Moreover, it is interactively and incrementally built up on the basis of the services available. In other words, the taxonomy is initiated with a reference taxonomy, e.g. CYC or Google directory, and then it is gradually tailored to the availability of services in LBS. This adaptive taxonomy building approach fits with the "dynamics" character of the LBS. The relations defined in chapter 3, such as (Discriminant) Is-A, Disjoint and Part-of, can be commonly used between services in the service classes taxonomy.

To tackle to the semantic heterogeneity of service profiles, the core ontology also serves as the semantic alignment authority in service module. When a service profile enters in LBS, its content will be parsed and further associated with certain service class in core ontology. To achieve this objective, different components associated with service module (mainly including the *service profile matcher*, *mapping library* and *service repository* as presented in Figure 4.2) must closely cooperate. Further, to ensure the consistency and inter-operability between the core ontology and service mappings, the mappings between service profiles and core ontology are also represented in ontologies.

4.1.3 Chapter Outline

This chapter takes a close look on the service module. We start with the introduction of service classes taxonomy and then give our definitions of certain properties/relations for service classes. Next, we explain the definitions of data profile and service profile at the syntax level, as well as their compatibility with SWS definitions in W3C. Finally, we present our approach of mapping service profiles against core ontology, maintaining the service profiles, and representing the mappings in terms of ontology.

4.2 Service Module

4.2.1 A Taxonomy of Service Classes

In the previous decades, people got used to looking up the yellow-pages to find out local services. By checking up the list, users can easily select the service in a certain category. It was also the initial prototype of current services web portals, such as Yahoo!local[7] or Google!Maps. Moreover, these web-based local portals enable users to find out a business within a specific geographical location or region. And the function and location of the services are indexed and organized by the search engines as *a priori*.

Similarly, the core ontology is based on an *a priori* taxonomy of services that provides a conceptual view of the domain independently of the idiosyncracies of heterogeneous data organization in the data sources. It serves as the conceptual layer to organize the services into hierarchies and to provide a general view about heterogeneous data profiles (i.e. data layer).

The taxonomy in LBS is an hierarchical taxonomy, i.e. a tree structure of classifications for the services in the core ontology. As the frame of the service classes, the taxonomy primarily starts with a set of top-level nodes (the ones immediately below the root node $\top$). And these top-level nodes represent a set of disjoint business functions, since services are usually denoted as: *rental, cleaning, selling, consulting, etc.* We consider that services in the same business function class potentially share the similar business mode, conditions and process. For instance, all 'rent' services may require a deposit or the users provide an ID card, and follow a process workflow like '*choose the product*' $\Rightarrow$ '*pay a deposit*' $\Rightarrow$ '*use the product*' $\Rightarrow$ '*return and pay*'. This disjoint business categorization can also help reducing the search space, as the business terms (e.g. rent, repair...) can be used to match the query. For instance, assume the query is *"rent a car"*. Analyzed by a query pattern, it fits the pattern "verb+noun", so that the business function is "rent" and its subject is "car". Further the query can be rewritten in the format which the core ontology can deal with, i.e. Rent $\sqcap$ $\exists$hasRentalProduct.car. Regarding the query formulation and processing, it will be discussed in detail in the Chapter 7 and 8.

After having the initial taxonomy, the LBS designer will study the locally available services, and enrich and modify the taxonomy. In our approach of building the core ontology, the test services data comes from the yellow-page. when assigning services to the initial taxonomy, it is potential that the taxonomy is unsuitable for classifying available services. In this context, the initial taxonomy needs to be modified (combined, added, or removed). For instance, some similar service classes can be combined into a single class, the less important

[7] http://local.yahoo.com/

properties can be regarded as optional ones. Services which do not exist in the initial taxonomy but often occur in queries should be searched and identified in local service profiles. In contrast, the service classes that have never been found in local service profiles should be removed.

In Figure 4.1, we show part of the service taxonomy in the core ontology and suggest our approach of organizing the services according to the business functions.

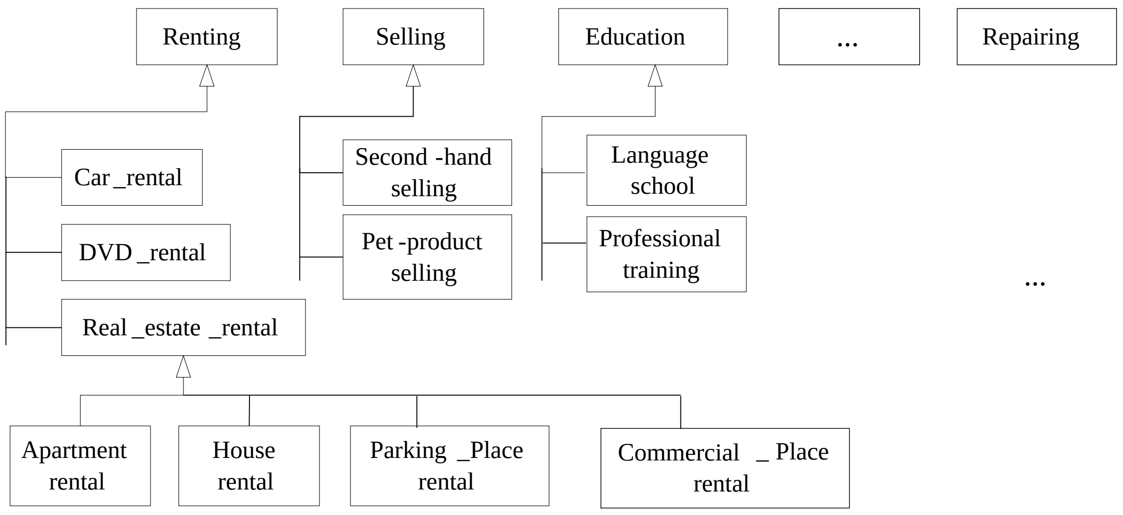


Figure 4.1: Part of the Service Taxonomy in our LBS core ontology.

4.2.2 Service Classes

In last chapter, we presented the definition of *Module Class* (see Definition 3.2). Following that definition, we can easily have the definition of *Service Class* by specifying the class belonging to *service* module:

DEFINITION 4.1. Service Class. *A Service Class c_s is a class in Service Module Classes C_s, defined as a triple (c, service, axioms) where c is the name of the service class,* service *specifies the module name of c_s, and axioms is the set of axioms that occurs in defining the class. (Refer to the axiom definition of Module Class in Definition 3.2)*

Our definition of service class is also compatible with the definition of *service profile* in OWL-S. *An OWL-S profile depicts a service as a function of three types of information: who provides the service, what function the service computes, and a set of features that specify characteristics of the service.* Similarly, our ”service class” provides an abstract and consistent description by being associated with a set of properties, e.g. mandatory properties as spatio-temporal properties or input/output property. Beyond OWL-S profile, our service class definition allows the explicit semantic definition in terms of ”axiom” and explicitly specifies the value of serviceCategory by referring its superclass(es) to class(es) in the service class taxonomy. The axioms provides the richer and disambiguating semantics than the description in OWL-S, by enumerating its individuals, defining its superclass and/or discriminating property(s).

4.2.3 Precondition and Post-effect Properties

In the last chapter, we presented certain attributes in the LBS. For instance, the *input property* specifies the necessary input parameter to execute a service for users, e.g. the property pickup-city for the *car rental* service. Input property information is provided by the query to the service. Without it, the service cannot properly operate. Execution of a service can also be subject to certain conditions being satisfied. As stated in OWL-S specifications, *if a process has a precondition, then the process can not be performed successfully unless precondition is true.* Preconditions rely on information that is not provided via input properties. The non-satisfaction of a precondition entails that the service should not be used, although it could operate correctly. We could say that input properties are inherent to the service, while preconditions restrict the context in which the service can be used.

As the execution of a service may influence the execution of another service, knowledge of preconditions is complemented with knowledge of post-effects. Intuitively, a post-effect is a change in the state of the world that is due to the execution of a service and that is significant with respect to the operation of other services. As discussed later in this chapter, preconditions and post-effects can be combined to organize and monitor ("orchestrate" in semantic web term) the execution of a group of services.

In OWL-S, the result of a service's execution is represented in terms of coupled *outputs* and *effects* in diverse conditions, i.e. under what condition, certain outputs and effects will be ensured to occur. Outputs refer to the information of target services acquired by the user, and effects mean the post-effects described above. In LBS setting, both precondition and post-effect are important properties to describe the service's functionality and potential results. We do not discuss the *outputs* property of a service, since the output of a service can be highly different. It largely relies on many factors, such as the display capability of the mobile device, constraints of OS or software embedded in the device, the details of the service profile, or the further interaction between the user and LBS. Rather than discussing further on services' automated discovery and orchestration, we are motivated by simplifying and explicating the preconditions and post-effects to end-users so as to enable users to understand the service's execution, precondition and potential results.

DEFINITION 4.2. Precondition Property. *A Precondition Property is an OWL objectProperty between a service class and an axiom, such as hasPrecondition(S, C), where S is a service class, and C is a module class or an axiom (⊓, ⊔, ¬ based class expression).*

For instance, hasPrecondition(Bar, AdultUser) where Bar is the service class, and AdultUser $\equiv$ User $\sqcap$ $\exists$hasAge.$\geq$18. In this example, the class AdultUser is a discriminated class, with the constraint on property hasAge of the class User. In addition, the precondition is distinguished from input property, since the service can still be discovered without knowing if the precondition is satisfied, but in that case, LBS can not ensure the success of the service execution. For instance, if the user is younger than 18-years old but does not specify that in her(his) user profile, the bar service can be found by LBS, but the user may finally fail to access to the bar if (s)he is unable to prove his/her age older than 18-years old.

The post-effect describes the corresponding changes of the state of the world after the service execution. As addressed earlier in this chapter, we do not emphasize the automated inter-operations between services, the introduction of post-effect property is mainly aimed at suggesting users with diverse conditions and their corresponding post-effects of a service. The post-effect can be particularly useful in expressing the multiple means of the service achievement. For instance, a visa can be obtained via sending as a registered courier by the visa authority, alternatively, it can also be received via picking up by the user herself (himself). Similar to precondition property, the post-effect property can be defined as follows:

DEFINITION 4.3. Post-effect Property. *A Post-effect Property is an OWL objectProperty between a service class and an axiom, such as hasPostEffect(S, C), where S is a service class, and C is a module class or an axiom ($\sqcap$, $\sqcup$, $\neg$ based class expression).*

For instance, to rent a car, when the user has chosen a car rented by a service provider, the user (id:uID) may be given a confirmation number. This confirmation number may be an input property use the cancelation functionality of the car-rental service. Thus, the confirmation number can be described as post-effects as follows: hasPostEffect(CarRental, ConfirmationNumber), where ConfirmationNumber is a number sent to the user to confirm the car-rental reservation and to enable to cancel the reservation using it. In the later part of this chapter, we will show how to apply the precondition and post-effect property to illustrate the orchestration relation of services.

4.2.4 Relations

Beyond the obvious Is-A relation (and its more precise variant, the discriminant Is-A relation) other relations are used to link service classes in the core ontology to make the search for services more efficient. An important one is the functional similarity relation, which allows a service to be taken as an alternative to another service. When for some reason the latter is not available, the former may be used instead. In a query answering strategy, using these relations, the LBS can achieve more successful searches (enlarging the set of potential answers to a query). Also useful to expedite search is the disjoint relation, which allows discarding irrelevant classes to more rapidly focus on the relevant ones. We also use transmission relations, which allow explicating the conditions under which a service class C_2 should be used instead of a service C_1. Finally, whole-part and orchestration relations support the organization and scheduling of complex non-atomic services.

Functional similarity relation. These relations are used to relate service classes that despite being different (not one subclass of the other) may offer similar functionality in a given context. It means for a certain query, services that are associated with the *functional similarity relation* can accomplish functionally similar requests. For instance, in Example 4.1, *bus* and *regional train* can offer the same public transportation function. If the request is concerning about 'bus', but bus service fails to fully satisfy the user's query (e.g. temporal constraint), in this case, its functional similarity class *regional train* service may replace the *bus service* to respond to user's request.

However, although functional similar service classes can alternate in some contexts, they still differ in some aspects represented by properties or conditions. When LBS recommends the service to users, these properties and conditions must be taken into account. For instance, consider the query 'the movie *Titanic* tonight', and three different services concerning *movie* offered in diverse formats: *cinema*, *DVD*, and *file download*. All of them can potentially satisfy the query, but they differ in the format of the movie entertainment, visual effect, price and extra requirements, as shown in Example 4.2. Using the *functional similarity* relation, LBS can relax the original query Q(A) by replacing the service concept A with another service concept B (A and B are two service classes defined in the same *functional similarity* relation), when the original query can not be satisfied. We give a formal definition on the functional equivalency relation and illustrate it with two examples Example 4.1 and Example 4.2.

DEFINITION 4.4. Functional Similarity relation. *A Functional Similarity relation is a relation on a set of classes, such as* $\simeq_{FS}(S,\ a,\ C)$, *where*

- *S is a set of service classes, such as $s_1, \ldots, s_i$ and $i \geq 2$;*
- *$\exists$ concept a, $\forall s_i \in S$, $s_i \sqsubseteq \mathsf{a}$, where a describes the common functionality of S;*
- *$\forall s_i, s_j \in S$, $s_i \neq s_j$ and $s_i \not\sqsubseteq s_j$;*
- *C is a set of axioms to describe extra conditions than its functional similar classes if they exist.*

Example 4.1. Bus vs. Regional train between *Lausanne* and *Renens*.
Similar Functionality: alternative city public transportations with same departure 'Lausanne' and destination 'Renens'.
Important Differences: journey duration, times of transit.
Condition: no special condition.

$\simeq_{FS}$ (tlr: public transport between Lausanne and its neighboring commune Renens)[8]:
$S_{tlr} = \{\text{RegTrain, Bus}\}$,
$F_{tlr} = \{\text{RegTrain} \sqsubseteq \text{PublicTransport} \sqcap \exists\text{Departure.}\{\text{Lausanne}\} \sqcap \exists\text{Destination.}\{\text{Renens}\}$,
$\quad \text{Bus} \sqsubseteq \text{PublicTransport} \sqcap \exists\text{Departure.}\{\text{Lausanne}\} \sqcap \exists\text{Destination.}\{\text{Renens}\}\}$,
$D_{tlr} = \{\text{RegTrain} \sqsubseteq \exists\text{TransferNo.}{=_0} \sqcap \exists\text{JourneyMinute.}{\leq_{10}}$,
$\quad \text{Bus} \sqsubseteq \exists\text{TransferNo.}{\geq_1} \sqcap \exists\text{JourneyMinute.}{\geq_{20}}\}$,
$C_{tlr} = \emptyset$.

In the above example, there are two functional similar classes RegTrain, Bus which serve as the public transport between city Lausanne and neighboring commune Renens (and other neighboring communes or villages or cities are possible as well, such as Lutry and Morges). From the description above, the relation can be also regarded as a view which helps LBS to provide the more suitable service when the exact matching can not achieve. For instance, assume the query has additional condition on temporal dimension, i.e. 'at 0:00am'. Since the bus can not provide any service between 0:00am and 6:00am, but the regional train service still runs on the route from Lausanne to Renens until 0:20am, the request can be satisfied with its functional similar counter-part *Regional train* service. In addition, when the user prefers the shorter duration or less transfer times, LBS would recommend the train other than the bus.

Example 4.2. Movie service: Cinema, DVD_rental or Download.
Similar Functionality: provide the movie entertainment.
Pre-condition: deposit is mandatory for DVD_rental and internet connection is required for movie download.

[8] The examples in this chapter are expressed in the DL/DL-Extension format as described in [BaaderKW05] of the DL handbook

Important Difference: the visual effects are different, and the modes to access to the movie service are different.
$\simeq_{FS}$ (movie):
S_{movie} = {Cinema, DVD, MovieDownload},
F_{movie} = {Cinema$\sqsubseteq$ EntertainmentService $\sqcap$ $\exists$Mode.{Movie},
DVD $\sqsubseteq$ EntertainmentService $\sqcap$ $\exists$Mode.{Movie},
MovieDownload $\sqsubseteq$ EntertainmentService $\sqcap$ $\exists$Mode.{Movie}},
D_{movie} = {Cinema $\sqsubseteq$ $\exists$VisualEffect.{Excellent} $\sqcap$ $\exists$ ScriptsLanguage.{French},
DVD $\sqsubseteq$ $\exists$VisualEffect.{Good} $\sqcap$ $\exists$ ScriptsLanguage.{French, English, German},
MovieDownload $\sqsubseteq$ $\exists$VisualEffect.{Good} $\sqcap$ $\exists$ ScriptsLanguage.{French}},
C_{movie} = {Cinema $\sqsubseteq$ $\exists$hasCondition.$\perp$,
DVD$\sqsubseteq$ $\exists$hasCondition.RentalDeposit,
MovieDownload $\sqsubseteq$ $\exists$hasCondition.InternetConnection}.

In the above example, we show the different but alternative modes of services on movie: Cinema, DVD and MovieDownload. Different from the example Example 4.1, this example shows the different properties and specific conditions of all services classes in S_{movie}. For the cinema service, it is featured by the *excellent* visual effect, and scripts-language is only in *French*, without any condition. Alternatively, the DVD service can provide *good* visual effect and offer scripts in *French, English, German*, but needs to pay the deposit as the condition of the service.

In this example, we do not consider the spatial and temporal constraints of all services. For instance, the cinema service must be accessed in a cinema during certain interval, the DVD_rental can be chosen within the open-hour of the DVD_rental_shop, and the download process takes certain time to obtain the whole film file etc. Regarding the query processing and service ranking, other context information becomes useful to determine what types of service is most appropriate for current user's request. It will be discussed in detail in Chapter 7 & 8 by taking into account all dimensional information, such as the temporal and spatial constraints in the query, user preferences and other contextual knowledge.

Transition Relation. Transition relation associates a set of functional equivalent classes which will transit from one to another if certain rule is satisfied. The transition can be symmetric or asymmetric. As shown in Example 4.3, when it is over 0:00 am, the normal bus service terminates, and it is replaced by a different service class, i.e. taxi-bus. The taxibus has the similar functionality, and possibly has same running route. But after 6:00am, the taxi bus terminates, and the bus services run again. The two service classes transit to provide city bus service following the certain temporal rules.

DEFINITION 4.5. Transition relation. *A Transition relation is a relation on a pair of classes, such as $\bowtie_T(s_1, s_2, r)$, where s_1 and s_2 are a pair of service classes, and r is an axiom to describe how the the transition happens from s_1 to s_2.*

Example 4.3. Bus vs. Taxi-bus.
Same superclass: city bus service.
transition rules: running time are complement over the lifecycle 24-hours. Bus service operates from 6:00am to 12:pm and taxi-bus operates from 0:00am to 6:00am. This transition is a bi-directional (or cycling) transition.

$\bowtie_T$ (24h-bus: Bus and Taxibus transition in 24 hours)
s_1 = Bus, s_2 = Taxibus,
r = {Bus $\sqsubseteq$ CityBus $\sqcap$ $\forall$RunningTime.Daytime,
TaxiBus $\sqsubseteq$ CityBus $\sqcap$ $\forall$RunningTime.$\neg$Daytime}

In this example, the transition relation between the bus and taxi-bus describes their complement roles in 24-hours bus services. Both of them can provide bus service to users, and they operate on complement intervals of a life-cycle, i.e. a day. This example shows how to correlate temporal complementary services, i.e. they transit from one to the other following temporal rules. Similarly, other types of transition relations between services may occur. For instance, baby-care services can transit to enfant-care group if the baby is older than 3-years. It can be noticed that the example of baby-care transition is not symmetric.

In addition, it is worthwhile explaining the difference between the *transition* relation and the *functionally similar* relation. The former one occurs between sibling service-classes, and there exists the transition rule, in the above example, the transition occurs on complement service time-intervals. But for the latter one, it is not necessary that the services in the relation are sibling service-classes, e.g. regional train and bus, and they are often similar (or equivalent) in the spatio-temporal characters.

Orchestration. It defines how a set of unit-services[9] can coordinate together to achieve the composite service. Beyond the aggregation relation, the orchestration is not only a simple binary relation between the whole and its units, further, it renders the sequential & conditional relation between the unit-services, i.e. coordination. In other words, unit-services must be carried on in a certain execution sequence (i.e. chronological order); for each unit-service, it can carry on if and only if all conditions (if one exists) have been satisfied. Another distinction between the orchestration and aggregation relation is that, the orchestration allows multiple ways of coordination. For instance, a service S can be achieved by orchestrating unit-services as $a \rightarrow b \rightarrow c \rightarrow e$ in this order; alternatively, S can also be achieved by orchestrating different unit-services set such as $a \rightarrow d \rightarrow f$. In the semantic web service community, many efforts have been made on automated semantic web services discovery and orchestration. To describe the *orchestration* relation in LBS, we refer to the definition of *location-based service flow* in [NSD+05] and present it as follows:

DEFINITION 4.6. Location-based Service Flow. *A location-based service flow with respect to some initial conditions P_0 is a finite sequence of services $\mathsf{SF}(P_0) = (s_1, s_2, \ldots, s_n)$ where for each service $s_i \in \mathsf{SF}$ (i=1...n), s_i = <d_i, p_i, e_i> (d_i is the description of service s_i, p_i is its precondition, and e_i is its post-effect.), all the following conditions hold:*

- *for s_1, $P_0 \sqsubseteq p_1$;*
- *for s_i, $i > 1$, $P_0 \sqcup e_1 \sqcup e_2 \sqcup \ldots \sqcup e_{i-1} \sqsubseteq p_i$;*
- *for s_i, $i > 1$, for each concept A occurring in e_i, $P_0 \sqcap e_1 \sqcap e_2 \sqcap \ldots \sqcap e_{i-1} \not\sqsubseteq A$.*

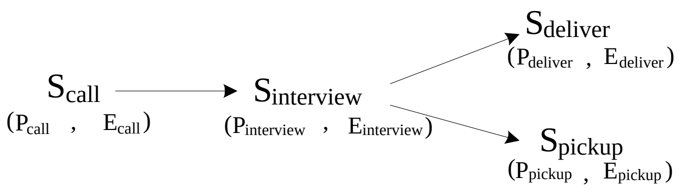
The Definition 4.6 describes the service flow composing of a set of services with a set of preconditions and post-effects. Let us describe the orchestration relation between a whole service and a set of service flows as follows:

[9] A unit-service refers to a service as a part of the whole service.

DEFINITION 4.7. Orchestration Relation. *An orchestration relation is a relation* $\oint$*(S, (*SF_1*, ...,* SF_i*)) where the service* $S = <D, P_0, E>$ *and a set of service flows (*SF_1*, ...,* SF_i*) (*$i \geq 1$*), where*

- *D is the service description of S,* P_0 *and E are respectively the precondition and post-effect of S.*
- *for each service flow* $SF_j \in (SF_1, \ldots, SF_i)$*, it holds* SF_j *provides the service S.*
- *for any two service flows* SF_a*,* SF_b*,* $SF_a \neq SF_b$*.*

Example 4.4. Visa Application Service.
Descriptions: The visa can be applied in two ways: 1) Call reservation → Interview → Visa Pick-up; 2) Call reservation → Interview → Visa Delivery.
$\oint^{Orch} = (S_{visa}, (SF_1, SF_2))$:
$S = S_{visa}$, $SF_1 = (S_{call}, S_{interview}, S_{pickup})$, $SF_2 = (S_{call}, S_{interview}, S_{deliver})$.



In the Example 4.4, the visa application can be achieved by following two possible service flows, i.e. for a composite service, there exists one or multiple service flows. Each flow is composed of respective unit-services with certain preconditions and effects. In this example, both share a common starting service (i.e. S_{call}) but have different ending services (i.e. $S_{deliver}$ and S_{pickup}). Since the issues of service composition and orchestration are not the main focus of our work, we will not go further on this topic. However, it suggests that existing work and definitions concerning SWS discovery and orchestration can be easily integrated into our work.

Condition. In the *Orchestration* relation, we introduced the term *condition*, which must be satisfied to proceed the next unit-service and eventually to achieve the whole service. In LBS, there is a specific relation between service classes, called *Condition Relation*. It represents that a service A can be accessed *iff* a service B has been successfully achieved by the user. It is different from the service orchestration since the classes involved in condition relation are independent service classes, and they do not intend to achieve the same objective as defined in Definition 4.7. For instance, beginning-level ski-course may be the condition of attending the middle-level ski-course. We defined the condition relation between two classes as follows:

DEFINITION 4.8. Condition Relation between Classes. *A Condition relation is a partial order relation* $\prec$ *on a set of classes such that* $c_1 \prec c_2$ *implies that* c_1 *is a necessary condition of* c_2*.*

In OWL, we directly define the condition relation between two classes as a global property *hasClassCondition*. In cases that the precondition is not applied to all instances of a class, the property is not total property and the cardinality will be set as "0" by specifying the property restriction owl:mincardinality. Accordingly, both range and domain of property *hasClassCondition* are classes, and any condition relation between two services is defined as an instance, and this property is neither symmetric nor reflexive.

4.3 Data Profile and Service Profile

We assume all service descriptions come from a set of data profiles. A data profile describes a data source that either directly provides services or has complete knowledge about who can provide the services in certain domain(s). In other words, a data profile describes a service provider or a service portal. For instance, as shown in Example 4.5, a car service provider presents its basic information and a set of sketchy service descriptions in its data profile. Alternatively, a data source can hold the information on a set of services from different providers, e.g. the cinema web-site[10] provides the recent movie information of all cinemas within Lausanne city. However, the data profile only needs to present the basic information about its services, rather than all intensional data. For instance, the data profile of a book selling service may only present its function, i.e. selling book, or in more details, the categories of books (e.g. text book, children books etc), but the information such as the book names and ISBN are kept at the data source.

DEFINITION 4.9. Data Profile. *A data profile is a tuple describing a data source, such that D = (ID, <P, I>, S), where ID is the unique identifer of the data profile D, P is a set of properties of the data source, I is the corresponding instances of the properties P, S is a set of service profiles such as S =*$(s_1, \ldots, s_i)$ *and* $i \geq 1$.

Example 4.5. A data profile. In this example, we present the data profile of a car agency. It contains the mandatory properties such as *name*, *location* and *open time*. Definitely, the non-functional properties of the data profile are not limited to these ones. From the data profile, we also know that this car agency provides three types of services: car rental, car repair and car accessory.

Data Profile: Car agency 1:
- name: La Roche Agency
- location: Av. de la Confrérie 19, Lausanne.
- open time: everyday.
- service:
Service 1. car_rental.
- deposit: 300 CHF - 400 CHF
- price: Honda(100CHF/day), Benz(150CHF/day), Smart(100CHF/day).
Service 2. car_repair.
- specialized mark: Benz.
- number of technicians: 4.
Service 3. car_accessory
- CD and radio.

With the help of the service profile mappings (see Definition 4.11), the service profile(s) can be aligned to the core ontology, and the information in the service profile will be added into the core ontology as instance of corresponding service class. In addition, the necessary information presented in the data profile, e.g. location, open-time, or information provider, will be added to the corresponding instance in the core ontology. Example 4.7 briefly presents a data profile together with one of its service profiles. From the Example 4.7, we observe some features of the service profiles: 1) a service profile often inherits a set of

[10] http://lausanne.cinemas.ch/home.php

basic information from its data profile such as identifier (or name), service provider, location and open time; 2) service profiles can present different properties, e.g. *car type* and *car mark*; 3) service profiles may present data values of the same property differently, i.e. in different data types, and/or at different levels of abstraction. For instance, 'deposit' property has value *yes* or *a value range* (300-400 CHF) in two profiles. We give the formal definition of service profile as follows.

Example 4.6. Service profiles. In this example, we present two *car rental* service profiles. They may be from the same data source specialized in car rental service information, or from different data sources.

Car_rental 1	**Car_rental 2**

service provider = La Roche Agency;
location = Av. de la Confrérie 19, Lausanne;
open time = Monday - Saturday;
deposit = 300 CHF;
car type = Diesel, petrol vehicle;
car mark = Honda, Ford, Toyota, Fiat;
price = Honda(100 CHF/day), Ford(120 CHF/day), Fiat(100 CHF/day), Toyota(80 CHF/day).
	service Provider = La Vinci Agency;
	location = Av. de Morges 118, Renens;
	open time = (Monday-Friday) 9:00-19:00, (Saturday) 8:00-18:00, (Sunday) close;
	deposit = (EuroCar-member) No, (non-EuroCar-member) Yes;
	car type = car, truck, coach.

DEFINITION 4.10. Service Profile. *A service profile is a data tuple describing a single service, such that S = (sID, <P, I>), where sID is the identifier of the service profile S, P is a set of properties of S, I is the corresponding instances of the properties P.*

Example 4.7. Example Specification of the service profile *Car_Rental 1*: It firstly presents the schema of the service profile in OWL, then gives the facts on the concepts and properties.

```
<owl:Class rdf:ID="SProfile">
    <rdfs:subClassOf>
    <owl:Restriction>
    <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasServiceCategory"/>
    </owl:onProperty>
    <owl:mincardinality rdf:datatype="&xsd;nonNegativeInteger">
        1</owl:mincardinality>
    </owl:Restriction>
    </rdfs:subClassOf>
    ...
</owl:Class>
<owl:DataProperty rdf:ID="hasServiceCategory">
    <rdfs:domain rdf:resource="#SProfile"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DataProperty>
<owl:ObjectProperty rdf:ID="hasAddress">
    <rdfs:domain rdf:resource="#SProfile"/>
    <rdfs:range rdf:resource="#SP_Address"/>
    ...
</owl:ObjectProperty>
```

```
<owl:DatatypeProperty rdf:ID="hasOpenTime">
    <rdfs:domain rdf:resource="#SProfile"/>
    ...
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="hasCarManufactor">
    <rdfs:domain rdf:resource="#SProfile"/>
    <rdfs:range rdf:resource="#SP_CarManufactor1"/>
</owl:ObjectProperty>
<owl:Class>
    <owl:oneOf rdf:parseType="Collection">
        <owl:SP_CarManufactor1 rdf:about="#Honda"/>
        <owl:SP_CarManufactor1 rdf:about="#Ford"/>
        <owl:SP_CarManufactor1 rdf:about="#Toyota"/>
        <owl:SP_CarManufactor1 rdf:about="#Fiat"/>
    </owl:oneOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasCarType">
    <rdfs:domain rdf:resource="#SProfile"/>
    ...
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasDeposit">
    <rdfs:domain rdf:resource="#SProfile"/>
    ...
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasPrice">
    <rdfs:domain rdf:resource="#SProfile"/>
    ...
</owl:ObjectProperty>
...
<SProfile rdf:ID="Car_Rental_1"/>
    <hasAddress rdf:resource="#Address_Car_Rental_1"/>
    <ComplexAddress rdf:ID="Address_Car_Rental_1">
        <hasPostCode rdf:datatype="&xsd;int">1004</hasPostCode>
        <hasStreet rdf:resource="#Av.de_la_Confrrie_19"/>
    </ComplexAddress>
...
```

NOTE: the set of properties P describes the characteristics of a service, it includes the necessary properties of a service, such as its spatio-temporal availability, the service category (i.e. business function), and the condition to access the service. It may also contain some specific properties of the service; I is a set of instances over properties P, for each property p_j, if p_j is necessary, its corresponding value instance $i_j \neq$ NULL. In the Example 4.6, both services belong to 'car rental' service class in the Core Ontology, but they have different properties such as *car mark*. The same property *car type* renders diverse semantics in two profiles, i.e. fuel-based (i.e. {Diesel-driven car, Petrol-driven car, ...}) and Function-based (i.e. {car, truck, coach, ...}).

4.4 Service Profile Matcher

Semantic matching is a major problem in large-scale distributed data management, its applications range from schema integration, ontology matching, e-commerce, to semantic query processing and currently emergent web-based services management. Its relevant works encompass *schema (or ontology) alignment, merging, articulation, fusion, integration, evolution, and so on.* In [RB01], authors define *Match* as a fundamental operation in the manipulation of schema information, *"which takes two schemas as input and produces a mapping between elements of the two schemas that correspond semantically to each other"*. Similarly, Kalfoglou et al. [KS03] describe the *ontology mapping as the task of relating the vocabulary of two ontologies that share the same domain of discourse in such a way that the mathematical structure of ontological signatures and their intended interpretations, as specified by the ontological axioms, are respected.*

In LBS setting, naturally, semantic matching is a challenge of LBS data management because the heterogenous data services maintain and represent their data in an autonomous and independent way. As suggested in Figure 3.1, *Service profile matcher* plays the important role of conveying and unifying the data semantics between the core ontology and heterogeneous data profiles in LBS. The "matcher" actually performs two tasks: 1) to match and fuse heterogeneous service profiles into the core ontology; 2) in return, at the query processing phase, to transform the query's semantics in terms of core ontology to that of source service profiles. In addition, while a service profile is matched to the core ontology, the core ontology may require to be updated and/or extended. Accordingly, the changes of the core ontology potentially propagate to the evolution of the dependent mappings. In Figure 4.2, we briefly describe the architecture of the *Service Profile Matcher* and illustrate the matching process.

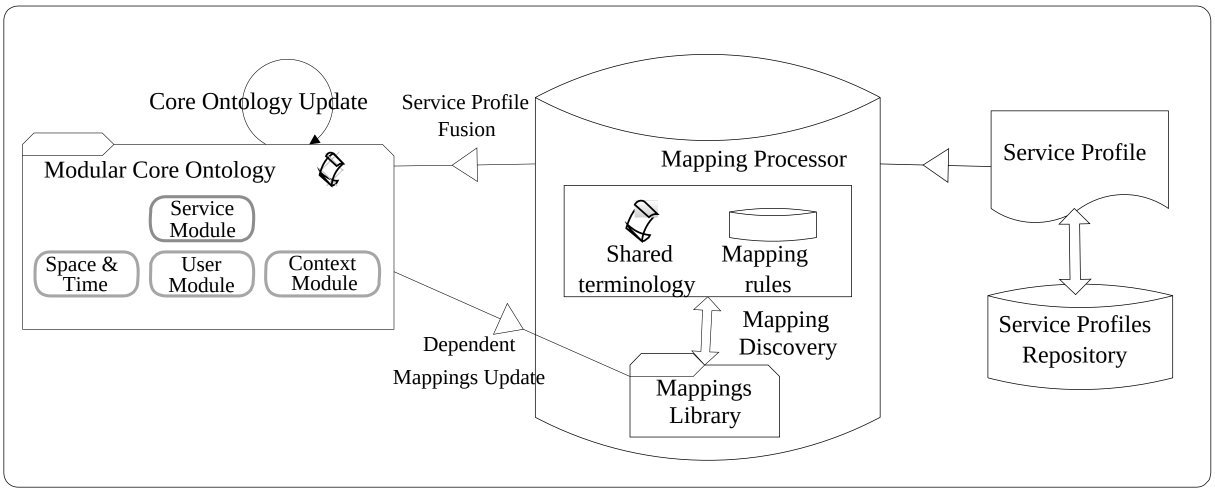


Figure 4.2: The Basic Architecture of Service Profile Matcher

Initialization of the Service Profile Matcher. Initially, both the *mapping library* and *service profile repository* are empty. The mapping discovery tool(s) and/or a set of mapping rules may exist in the mapping processor. In literature, researchers and practitioners explored various approaches and techniques to lift and normalize, map the words at the syntactical, structural and language levels, e.g. [VJBCS97]. The issues on how to discover the mappings between the service profile and core ontology are beyond the scope of this dissertation.

Update of the Service Profile Matcher. As LBS enters into use, whenever a single service profile is uploaded to *Service Profile Matcher*, each element in the service profile will be identified by the mapping processor. The elements' similarity measures may be computed considering the linguistics, taxonomy and property. Then it will recommend a list of mappings to relate the element in the service profile with the core ontology. The identification of the mappings might be (semi-)automatically achieved and/or supervised by the LBS experts. While the service is completely fused into a service class of the Core ontology, the service profile will be stored in the *Service Profile Repository* and their identified mappings between service profile and Core Ontology will be added to the *Mapping Library*.

Under some circumstances, the structure of target service class in core ontology may not suit for needs of real service profiles. For instance, in the core ontology, service class "restaurant" has a property "address" whose value constraint is string. However, in most "restaurant" service profiles, property "address" is defined as a complex property composed of "city" and "street". In this case, in order to make efficient the profiles matching as well the query processing, it is recommended to modify the structure of target service class of the core ontology accordingly. This modification may have a determinant effect on the dependent mappings in the mapping library. The *dependent mappings* denote a set of mapping instances in the *mapping library*, which associate an evolved element of the core ontology as the evolution of the core ontology. For instance, in the example above, all mapping instances including the property "restaurant->address" under the class *restaurant* are dependent mappings when property "restaurant->address" is changed as a complex property from a simple property. It functions in a similar fashion as the trigger in the database. Thus, the corresponding update of the dependent mappings is necessary and the updated mappings will replace the old ones in the mapping library. In addition, the unsuitable definitions of cardinality or property constraint at the initial design can also result in the evolution of the core ontology.

In last decade, researchers and practitioners from database & ontology community proposed diverse approaches of schema matching or integration and produced a large amount of results. The schemas include the relational database schemas, XML and compatible schemas (e.g. DTD and XSD), and OWL-based ontologies. In [RB01], authors provide a comprehensive survey on existing approaches of schema matching and classify them into schema-level and instance-level, element-level and structure-level, language-based and constraint-based. Rather than creating a new matching approach or optimizing the algorithms, we concentrate on employing existing approaches to deal with the semantic matching issues in LBS.

In the remaining of the section, we set out with the representation and manipulation of *mappings* based on the architecture presented in Figure 4.2. We also explain how to match and fuse a single service profile to the core ontology with the mappings. Then issues on core ontology evolution and mapping update are investigated. Issues on how the mappings can help to transform and process the query will be discussed in detail in chapter *Semantic Query Processing*.

4.5 Discovery and Representation of Mappings

In the literature, there exist diverse definitions and representations on schema mapping and ontology mapping. From the database perspective, Rahm and Bernstein, in [RB01], defined the *match* operation as a function that takes two schemas S1 and S2 as input and returns a set of *mapping elements* between two schemas as output. Each mapping element indicates that certain elements of schema S1 are mapped to certain elements in S2. Due to the specifics of the representations of schemas, (i.e., ER model, object-oriented model, XML model) different types of mapping relations, functions or expressions between schema elements can be chosen and defined.

Regarding the existing mapping approaches, the mapping expressions can be generally classified into two types: schema-level mappings and instance-level mappings. At the schema-level, the mappings can be further classified as name-based, constraint-based, and structure-based. The name-based mappings often employ the linguistics relations/functions between the elements, such as equality, synonym and hypernym etc. The constraint-based mappings are based on the exploitation and analysis of the structure of the database schema, such as the is-a relation, relationship cardinalities, and referential constraints. Structure-based mappings are applied by comparing the neighborhoods of the two elements separately in their schemas. Regarding the instance-level mappings, they generally fall into two categories: text-oriented and constraint-oriented. The former mainly apply Bayesian learners to find the relevant texts in the instance value; the latter discover the consistency between the domain values, character/numerical data patterns, even the distribution and average. These mappings are usually implemented by employing additional programming languages like Java or C.

Even though the database schema and ontology share many commonalities from the data management perspective, they distinguish from each other in terms of logic expressiveness, the delimitation between class and instance, the approaches of dealing with the data and structure evolution etc. as investigated in [NK04]. In particular, e.g. in OWL, specific properties and axioms such as functional-properties, inverse-properties, domain and range constraints, property constraints all need to be taken into account during the mapping process. This makes the ontology mapping more complicated than the schema mapping. Moreover, the way in which ontologies represent and reason over the data makes ontology matching different from the schema matching, so that the feasible approaches to representing and manipulating ontology mappings can differentiate from those of schema matching. In [Noy04], the author outlines three types of approaches to representing mappings between ontologies:

- to represent mappings as instances in an ontology of mappings, e.g. MAFRA framework [MMSV02], work done in [CM03], and definition in [ES05];
- to define bridging axioms in first-order logic to represent transformation, e.g. OntoMerge system [DMQ03];
- to use views to describe mappings from a global ontology to local ontologies, i.e. GAV(global-as-view)/LAV(local-as-view), e.g. OIS framework [CGL01].

Figure 4.3 provides the generic view of the semantic bridging ontology in MAFRA, where the mappings are defined as a set of instances of *semantic bridges* in an ontology. It is composed of three basic types of *entities* (i.e. Concepts, Relations and Attributes), the class *Semantic Bridge* relating the source entity and target entity, the class Service describing the transformation resources, the class *Rule, Transformation, Condition* which are used to specify the transformation-relevant information or constraints, and the modeling primitives *Composition* and *Alternative* used to support the relations over the semantic bridges, such as composition and mutual exclusiveness.

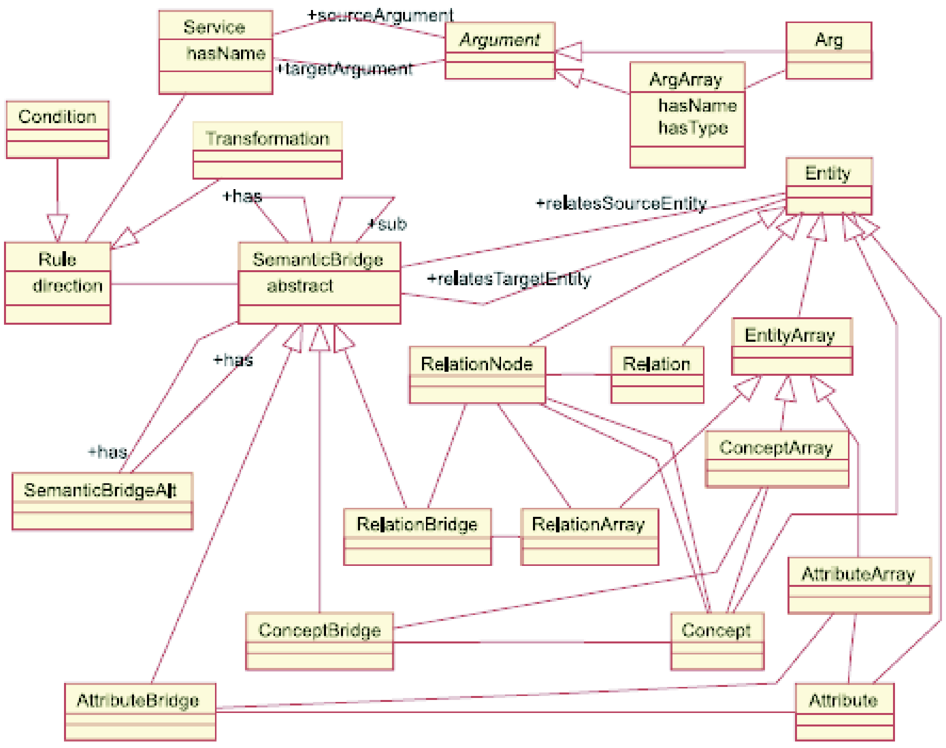


Figure 4.3: The Bridging Ontology view in UML. (i.e. Fig. 3 in [MMSV02])

Motivated by the mapping representations in MAFRA, we describe the mappings between the ontology and service profile in a similar way, i.e. any mapping can be represented as an instance in the mapping library, and two elements are separately from the core ontology and the service profile (see Definition 4.11). Different from the specifications in MAFRA, we do not specialize the mappings according to the types of the elements such as *concept-bridge*, *relation-bridge* and *attribute-bridge*. The elements involved in the mapping can be of type concept, property, relation, or individual, and can be either a single element or a complex one. Further, we allow two elements associated in a mapping relation to be of different types. For instance, the source property *car type* (i.e. property in service profile) is a simple property, and its value domain is constrained as an enumerated set {Diesel, Petrol}. However, the target property (i.e. property in the core ontology) can be a complex property, e.g., a property with multiple representations such as {Diesel, Petrol || Car, Truck, Coach}. In OWL, the complex property is generally described as a concept, as discussed in the Section 3.2.4. In this case, we need to map an element of property type to an element of concept type. The mapping also describes certain relations or functions between two elements. A typical one is the function 'Equal-to'(=) or 'Similar'($\simeq$). Other types of relations are also possible and have been investigated in literature, such as concatenation, aggregation, decomposition, etc. ([CGL01] [KS03] [Do06]). In a recent review [Noy04], the author classifies the approaches of ontology-mapping discovery into two categories: 1) to use the shared ontology to make a common grounding for knowledge sharing between ontologies; 2) to apply some heuristics and machine learning techniques to find the mappings. In the infrastructure of service profile matcher, we allow the various or combined deployment means of discovering mappings. We give the generic definitions on mapping library and mapping as follows:

DEFINITION 4.11. Mapping. *A mapping is a tuple describing a relation or function over two elements such that* $\mathcal{M}$ = *(id,* e_s, e_t*, MAP) where id is the unique identifier of the mapping instance,* e_s *is the source element from a service profile,* e_t *is the target element from the core ontology, and MAP is the binary relation between* e_s *and* e_t*, and* $\{\equiv, \cong, \sqsubseteq, \sqsupseteq, \textit{UNION}\} \sqsubseteq \textit{MAP}$.

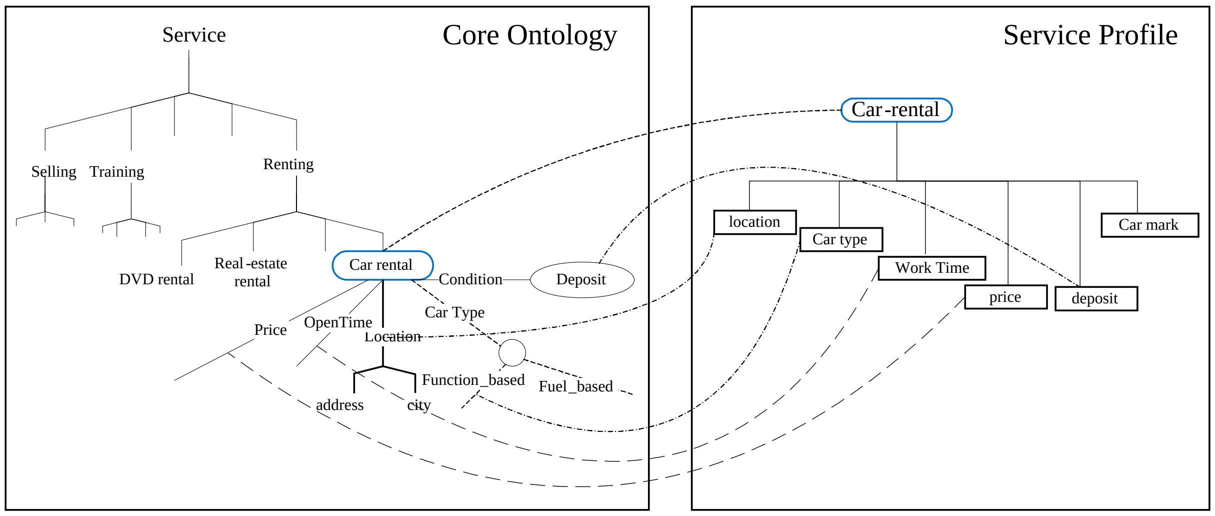
In OWL, the mapping $\mathcal{M}$ is defined in terms of class, elements e_s, e_t and identifier *id* are separately defined as its mandatory properties. For different classes of mappings (i.e. service mapping, property mapping and individual mapping), there exist diverse constraints on the cardinalities, domains and ranges of e_s, e_t. In the following OWL definition, we suggest the constraints on the cardinality of the property *hasSourceElement*, and the constraints on *hasSourceElement* can be defined in a similar way.

```
<owl:Class rdf:ID="Mapping">
<rdfs:subClassOf>
    <owl:Restriction>
    <owl:onProperty>
       <owl:ObjectProperty rdf:ID="hasSourceElement"/>
    </owl:onProperty>
    <owl:mincardinality rdf:datatype="&xsd;nonNegativeInteger">
        1</owl:mincardinality>
    </owl:Restriction>
</rdfs:subClassOf>
 ...
<owl:ObjectProperty rdf:ID="hasSourceElement">
    <rdfs:domain rdf:resource="#Mapping"/>
    ...
</owl:ObjectProperty>
...
</owl:Class>
```

DEFINITION 4.12. Mapping Library in LBS. *A Mapping Library describes the mappings between the core ontology and the service profiles, such as* $\mathcal{ML} = \{\mathcal{M}|m_1, \ldots, m_i\}$.

$\mathcal{M}$ is the mapping defined in Definition 4.11, $\mathcal{ML}$ is a union of all mappings defined in LBS. According to the target elements of mappings, the mappings can be categorized into service mappings (i.e. class-level), property mappings and individual mappings.

Example 4.8. An example of the mappings between a 'Car rental' service profile and core ontology. In the figure below, we illustrate part of the core ontology, a service profile *Car Rental*, and some of the mapping between them.



1. Service Mapping. As shown in the Example 4.5, the service *Car-rental* is mapped to the service class 'Car rental' in the core ontology. The mapping can be described in terms of a tuple such as $\mathcal{M}^{\equiv}_{Service}$ = (id, e^{svc}_s, e^{svc}_t, `Equal`), where e^{svc}_s is the whole service '*Car-rental*', e^{svc}_t represents the service class '*Car rental*' in core ontology, and `Equal` ($\equiv$) declares the 'Equivalent' relation between e^{svc}_s and e^{svc}_t. Because in this mapping the source element represents the whole service profile, this type of mapping is called *Service mapping*, denoted as $\mathcal{M}_{Service}$. When the mapping ontology is encoded in OWL, we define a super class `Mapping`, which has sub-types such as `ServiceMapping` defined below (i.e. the mapping $\mathcal{M}_{Service}$ aforementioned). The relation `Equal` is also sub-type of relation `Map` of the Definition 4.11. Further, we can make more constraints on the SourceElement and TargetElement beyond its super type as defined below and the axiom specifies the *equivalent* mapping between two elements. However, OWL provides a built-in axiom *owl:EquivalentClass* to represent two equivalent classes. It seems a straightforward way to represent the equivalent relation between two elements. However, certain advanced mapping relations, such as aggregation, decomposition, can not be directly represented by using OWL built-in axioms and require external transformation functions.

```
<owl:Class rdf:ID="ServiceMapping">
    <rdfs:subClassOf rdf:resource="#Mapping" />
    ...
</owl:Class>
 ...
<owl:ObjectProperty rdf:ID="hasSourceElement">
    <rdfs:domain rdf:resource="#ServiceMapping"/>
    <rdfs:range rdf:resource="#SProfile"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasTargetElement">
    <rdfs:domain rdf:resource="#ServiceMapping"/>
    <rdfs:range rdf:resource="#Service"/>
</owl:ObjectProperty>
<ServiceMapping rdf:ID="ServiceMapping_Car_Rental1">
    <hasSourceElement rdf:resource="#Car_Rental1"/>
    <hasTargetElement rdf:resoucrce="#CarRental_xxx"/>
 </ServiceMapping>
<SProfile rdf:ID="Car_Rental1">
<CarRental rdf:ID="CarRental_xxx">
 ...
```

Axiom on $\mathcal{M}^{\equiv}_{Service} = (id^c,\ e_s^{svc}, e_t^{svc},\ \texttt{Equal})$:
$\Rightarrow$ ((hasSourceElement, $?e_1$, $?m_1$) and (hasTargetElement, $?e_2$, $?m_1$) and ($\in$, $?m_1$, $\mathcal{M}^{\equiv}_{Service}$))
exists $(e_1 = e_2)$

Different from the generic mapping in existing approaches, the relation in our mapping can render certain semantics. More often, relation *Equal* is an ideal one between two classes since there usually exists some difference in the structure or constraints. In the above example, we just illustrate the mapping between service class and service profile, and relation *Equal* indicates both have equivalent service functionalities, rather than merely the *equivalent* structures and terminologies between two classes. Similar to some generic mapping relations, other relations such as subset($\subset$), overlap, are common in LBS.

2. Property Mapping. Consider a class being characterized by its logical definition, properties with other classes/individuals, and text descriptions. Obviously, the properties of a service class are crucial to describe the service class, and to suggest relations with other services. In LBS, a service profile describes a single service class without offering much information on the role hierarchy with its super-types/sub-types or the relation with other service classes. Therefore, property-mappings that we will discuss mainly fall into one of the these two categories: dataProperty mappings or alternative dataProperty mappings (i.e. certain complex properties, e.g. the property *address* in the core ontology is composed of two simple dataProperty 'address' and 'postcode'). Similar to the service mapping, the property in the service profile will be mapped to the property of the corresponding service class. For instance, the property *Work Time* has the same meaning and property constraint with property 'OpenTime' in the class *CarRental*. In this case, the property *Work Time* can be directly converted into 'OpenTime', and the mapping can be written as $\mathcal{M}^{\equiv}_{Prop}$ = (id, e_s^{prop}, e_t^{prop}, $\texttt{Equal}$).

However, it is very common that the property in the service profile is inconsistent with the target property in CO, the inconsistency may occurs in terms of the structure, the value domain or the constraints. In that case, it is necessary to either transform e_s^{prop} to the syntactic of e_t^{prop}, or tailor e_t^{prop} to e_s^{prop} if necessary. For instance, in Example 4.5, e_s^{prop} *location* needs to be decomposed into two simple properties, i.e. 'address' and 'postcode'. In return, when e_s^{prop} is a complex property and e_t^{prop} is a simple one, it is possible to aggregate the simple properties of e_s^{prop} to the format of e_t^{prop}.

$\mathcal{M}^{\rightleftarrows}_{Prop}$ = (id, e_s^{prop}, e_t^{prop}, $\texttt{Transform}$)
e_s^{prop} = Car_Rental1.location
e_t^{prop} = CarRental.location
= CarRental.location.address + CarRental.location.postcode

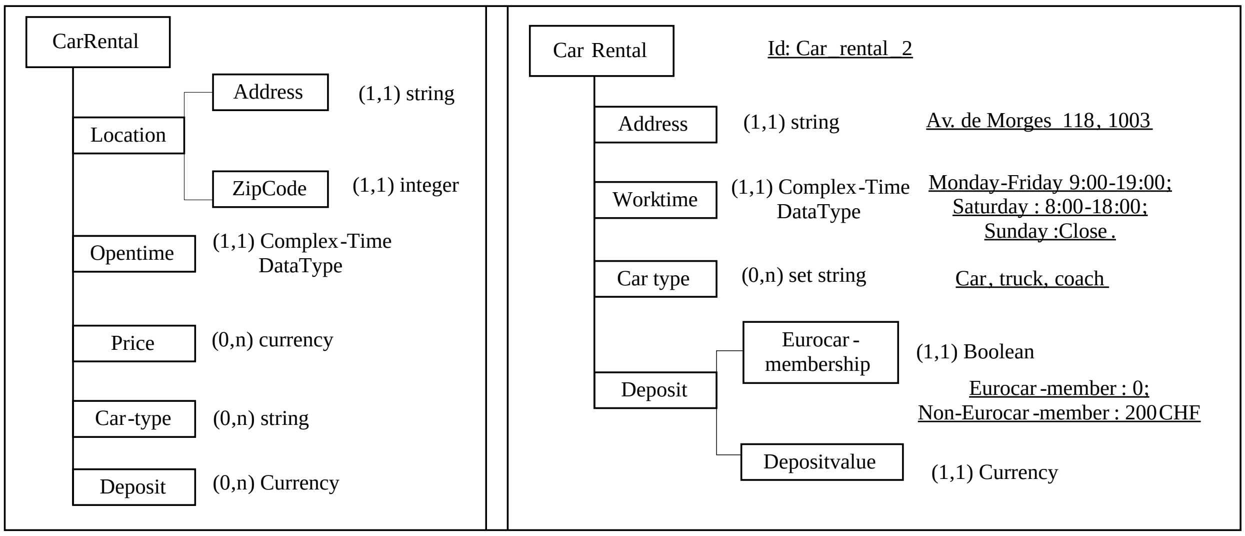
In the above example, when we transform source property 'Car_Rental1.location' to two simple properties, it is optimistic to keep the property-ranges (as well the constraints) of the source and the target properties consistent. For instance, the *postcode* information can be included in e_s^{prop} *Car_Rental1.location* in terms of string, but *postcode* in e_t^{prop} explicitly specifies its property-range as non-negative integer. To transform between different data-types, e.g. string $\rightleftharpoons$ integer, extra programming are needed.

However, in some circumstances, it seems difficult to represent the mappings (or functions) when the structures or the constraints of source/target properties are inconsistent. For instance, in the Example 4.7, for the service Car_Rental2, property *deposit* is dependent on the user-relevant information 'EuroCar-Membership' and has property-range 'boolean datatype' (i.e. 'Yes/No'); for the Car_Rental1, property *deposit* is a dataProperty with property-range 'currency'. In this example, we can find out the property-ranges of two source properties are distinguished in rendering their own semantics on the same property. In literature, there are mainly two ways to reconciling the inconsistency in structure and constraints: 1) to keep alignment with a target (or standard) property, but probably with the risk of losing some information of the source property, 2) to allow multiple representations on the same property, which often applies to transform the sources to a single data-warehouse [BR00] or to represent the object/property in a multiple-representation database [PSZ06]. In our work, two approaches can be combined to fit for different situations. For instance, for the property 'deposit', it is possible to specify it as a multiple-representation property. But for the property *open time* in Car_rental2, it needs to be aligned with the target property *OpenTime* since *OpenTime* is a mandatory property and needs to be in a unique format specified by LBS designer.

3. Individuals Mapping. In the database mapping, the individual mappings are important phase to completely import the source instances to the target database. When schema-level mapping is achieved, it is common to make the element-level mappings, e.g. two records separately from two relational tables. In our scenario, when a service profile maps to a service class in the core ontology, it is necessary to establish a link assigning a service profile to a service class in the core ontology. In occasional cases, it is possible that for a real-world service instance, there exist two different service profiles. The term *different* does not only mean they are from different data profiles, but also suggests two individual service profiles potentially have various structures and representations. Therefore, as a service profile is input to the LBS, it will firstly be identified and recognized if it is the same individual service described by another individual service profile already in the LBS. The approaches to evaluating the equivalency of two individuals can be diverse, but we mainly concentrate on what LBS will proceed once the equivalency between individuals is found. Similar to the solution of resolving the heterogeneity between two properties, to handle the individuals in heterogenous representations, one is to align it with a standard one, (i.e. integrate all properties in two service profiles into one individual service); the alternative is to keep multiple representations individually, but to link them as *equivalent* service individuals using axiom. The following assertions describe the mapping between two individuals, where one is from *CarRental* class in core ontology and the other is from the *Car_Rental1* service profile. As stated before, while a service profile is identified as a new service instance for LBS, it will be fused to the core ontology and a service profile instance will be inserted to the core ontology such as *#CarRental_123*.

```
    <owl:Class rdf:ID="IndividualMapping">
        <rdfs:subClassOf rdf:resource="#Mapping"/>
    </owl:Class>
    <IndividualMapping rdf:ID="IndividualMapping_1">
        <hasSourceElement rdf:resource="#Car_Rental1"/>
        <hasTargetElement rdf:resource="#CarRental_123"/>
    </IndividualMapping>
    ...
   <SProfile rdf:ID="Car_Rental1">
   <CarRental rdf:ID="CarRental_123">
```

Example 4.9. Class *CarRental* in CO vs. a Service Profile *Car_Rental2*. In this example, we simply illustrate how to insert a service instance in the LBS and build up corresponding mappings in ML. The fourth operation shows a series of operations in Core ontology and mapping library and consequent update of all relevant property mapping instances to this property evolution. The left part of Figure 4.5 refers to the metadata of the service class CarRental in the core ontology, and the right part represents a service instance.



1. Add a service instance in LBS.

m1 = (Car_rental_2, CarRental_102, Equal)
⇒ Create a sevice mapping m1 in mapping library ML,
⇒ Associate service profile CarRental_102 with class CarRental in CO.

2. Decompose a simple property *address* to a composite property *location*.

m2 = (address, location, decompose)
⇒ Add property mapping instance m2 to mapping library ML,
⇒ Associate value Av. de Morges 118 with CarRental_102's *location.address*,
⇒ Associate value 1003 with CarRental_102's location.zipcode.

3. Directly insert a property.

m3 = (worktime, opentime, Equal)
⇒ Add property mapping instance m3 to mapping library ML,
⇒ Associate value *Monday-Friday 9:00-19:00; Saturday: 8:00-18:00; Sunday:Closed* with *CarRental_102*'s *Opentime*.

4. To modify the class CarRental 's simple property to a multi-representation property, insert the new property mapping and its value in ML, and update all relevant mappings to Deposit in ML.

m4 = ($Deposit_{Car\_rental\_2}$, $Deposit_{CarRental}$, Map)
⇒ Modify the property *Deposit* of class CarRental to a complex property,
⇒ The new property *Deposit* associates two properties *Europcar-membership* and *Deposit-value*,
⇒ Add constraints on the cardinality of Europcar-membership as (0,1),
⇒ Add constraints on the cardinality of deposit-value as (1,1),
⇒ Insert value (**Yes, 0**) to *CarRental_102*'s (Europcar-membership, Deposit-value),
⇒ Insert value (**No, 200**) to *CarRental_102*'s (Europcar-membership, Deposit-value),
⇒ Update all property instances of Deposit of class CarRental,
⇒ Update targetElement of all Deposit's property mapping instances of class CarRental in $\mathcal{ML}$.

4.6 Chapter Summary

The service module provides the semantic abstraction of services in LBS, i.e. the taxonomy of services, the definitions of services with a set of properties, the relationships with other services. In this chapter, we start with the discussion on the characteristics of web services and semantic web services. However, their definition and data infrastructure do not suit well for the LBS needs, e.g. "Locality, Mobility and Dynamics". This calls upon a new service data management strategy for LBS. Differently, the service module in our core ontology have three aspects of functionalities: 1) a service registry which organizes the services in a hierarchical taxonomy, 2) a service alignment authority which can help to identify the semantics of service profiles and further assign it to a certain service class in the core ontology, 3) it helps to relate services in some practical ways, e.g. to relate the functional similar services, to express the detailed processes inside a service. In addition, we explain one of the important components relevant to service module, i.e. the service profile matcher. It handles the semantic heterogeneity between service profiles and core ontology, and illustrates some manipulations on service profiles. Finally, we present the mapping ontology, which stands for the mappings between core ontology and service profile, ranging from service class and property to values. In following chapters, we will delve into the details on the data management of context and user profiles, and explain how to define the *"Connections"* between different modules, i.e. Connection between service module and context module, and Connection between service module and user module. In return, these connections will facilitate to achieve the context-awareness and personalization of LBS.

Chapter 5

Context Information Management

5.1 Introduction and Motivation

As stated in chapter 2, our work focuses on how LBS can provide mobile users with context-aware and personalized services in terms of information delivery and exchange. Semantic support for these services is rooted in the knowledge that the LBS can exploit. LBS knowledge should describe the characteristics of the user, the content available in the data sources, and the environment (in the broad sense) in which LBS/user interactions are embedded. Hence, context information, user profiles and data profiles play a crucial role in raising the quality of the services provided by the LBS. More importantly, it is the interrelations and interactions between these pieces of knowledge that build the knowledge substrate determining how semantically effective will the information services be.

A common philosophical assumption asserts that everything is context-dependent. Indeed, for example, the name of a person, often seen as an inherent and unchangeable characteristic of the person, is in fact context-dependent as the same person may have an official name but also a nickname in an email address book, a familiar name used at home by family members, an alias used to chat on Internet, a code name as a member of a group, etc. However, when downsizing the universe to the world that is relevant to some specific IT application, things like a user name can become context-independent, meaning that whatever the usage of the information within this restricted application world, the value for that characteristic will be the same. For a smart LBS, the assumption that everything is context dependent materializes in the fact that every user query is checked against context data to see if the query should be reformulated differently because of the additional knowledge extracted from the context repository. For example, the query for a shopping area with a Chinese restaurant needs knowledge about the local shopping areas, knowledge that would typically be stored in the current spatial context. Conversely, once the LBS has returned the name of a suitable restaurant, the query whether this restaurant is open on Wednesday may be directly evaluated without looking at context data, because the concepts of open-days and Wednesday have a unique interpretation within the LBS. Given that LBS are characterized by the diversity of users and the unpredictability of their requirements, how to best understand users' intention thanks to the explicit/implicit context relevant to the query, how to represent and manipulate the context, and how to apply the pieces of context information in information selection become the new challenges for emergent context-aware services in general and for LBS in particular.

In addition, as discussed in chapter 2, *context, user profiles*, and *service profiles* are separate but closely interrelated parts in LBS. Especially, context often serves as the base and pivot to link and elaborate other parts so as to achieve the objective of context-awareness and personalization in LBS. For instance, user preferences may be context-dependent, such as "favorite sports can change from summer to winter" where both *summer* and *winter* are context concepts and the sports value will change according to the value of context class *season*. Services can also be regarded as context consumers because some services are sensitive to or even determined by some specific context, e.g. ski services being only available if there is sufficient snow.

In the sequel of this chapter, we firstly analyze the use, definition and modeling of context in the literature. Next we discuss how context data can be organized within an LBS, and propose and justify our approach to cope with the functionality of context in LBS. Finally, we present our approach of defining and manipulating the context information, and describing the relations and dependencies between contexts, with the motivation towards an adaptive and knowledgeable information selection.

5.2 Related Work

5.2.1 Context Definition

In the literature, the studies on context encompassed many significant research issues in diverse disciplines and applications, ranging from psychology, linguistic [BB05] and artificial intelligence, to nowadays context-aware computing [Dey01]. Generally speaking, in any open data management environment it is an essential issue to better understand what information the user is seeking before matching the request with the data. For example, an application domain that tends to become a major consumer of context data is web Search [Law00], where use of context is extremely relevant to the targeted goal of reducing the amount of pages returned to users while improving their relevance. In a static framework such as desktop web interactions, context may be determined by explicit user action, e.g. choosing a category (e.g. the results of 'hotel' in category 'travel' are different from those in category 'employment'), or by implicitly inferring contextual elements from the documents edited/browsed by the requesting user. However, in dynamically evolving computing environments, e.g. mobility frameworks, other types of context information driven from this dynamicity (e.g. user's location, activity, surroundings, etc.) become a crucial additional player in the information/service selection. This leads to the usual definition of context in LBS as below.

> *Context* in information services can be generically defined as any information which can determine or influence the selection of information to be delivered to the user in response to a given query (in a 'pull' style) or as a follow-up to some user's pre-specification (in a 'push' style).

To separate concerns, in our LBS framework we restrict the scope of context as only referring to information that describes the surrounding environment relevant to user's query, excluding user preferences and service profiles. Typical examples of context information in this sense include atmospheric data, traffic conditions, calendar data (including national and local holidays), and cultural and communication settings. This information depends neither on the user nor on the local service data. It is often available from local authoritative data sources, e.g. atmospheric data from the local weather broadcast agency. Context also includes higher-level information that can be inferred based on reasoning rules known to the LBS. An example is user's current activity (e.g. doing shopping, being at a meeting) which may indeed greatly influence the semantic interpretation of a query. Despite being related to the user, user's activity belongs to context (rather than to user's profile) as it is derived using a rule that is not user-dependent. Many efforts in reasoning high-level context have been made by using a logic-based system, e.g. first-order logic [RC03] or OWL-based reasoning [CF03] [WGZP04].

As LBS environments are not devoted to general purpose knowledge management, but to the specific task of providing local information extracted from available sources, context in LBS is restricted to hold information that may be useful in processing user queries, i.e. that can play a role in the match between user profiles and contexts (to determine which profile and which elements of a profile are relevant for the current context) and/or a match between context and the data profile (to be able to have a context-dependent selection of services).

5.2.2 Context Modeling

When targeting context-awareness, an immediate concern is how to encode and represent context information within the LBS framework. Context modeling and manipulation challenges include how to properly handle the many distinguishing characteristics of context data, i.e. their spatial, temporal, dynamic, distributed, interrelated, imperfect and ambiguous features [Dey01] [HI04] [vBFA05]. A well-defined conceptual model is needed to facilitate the development and evolution of context-aware services. Some abstract requirements for context models and subsequent evaluation of some family of models can be found in [SLP04]. We adopt a somehow more concrete view of the domain, as follows.

Context information in LBS describes what is the current status of the real world (sometimes called the current situation) when looking at it from a variety of perspectives. If there is a concern about time, for example, context information can inform about what time is it in the current time zone, which calendar system is in use in the current country, whether the current day is a weekday or a weekend day, etc. Context information can almost be everything, as everything can be perceived as potentially influencing human thinking and action, which is what we want to apprehend as precisely as possible in LBS. Context data is there to provide the current value of a selected variable information given the current context. This explains why earlier works on context simply represent it as a set of attribute-value pairs, the choice of the attributes being driven by the specific requirements of the targeted application (see e.g. [SAW94] for a definition of context data for environmental change management). The simplicity of this solution is counter-balanced by its poor expressive power, its ad-hoc basis meaning lack of portability, and its poor ability to support evolution. Basically, the solution does not support the functionality needed for smart context management, such as multi-level context definition or interrelationships among context data.

To achieve better expressiveness, a popular idea since has been to look at context data as forming a context database, and consequently use traditional conceptual data models for the description of this context database. Indeed, context data is a representation of some piece of reality, as we just pointed out, and that is exactly what databases are. Hence it is reasonable to plan using a database model to design the context database. Among the followers of this idea, Henricksen and Indulska have proposed an extended ORM (object-role modeling[1]) approach, in which context information is constructed as a set of objects (e.g. person, device, and communication channel for a message delivery scenario), each one described by its attributes and possibly linked to another object via binary associations. Their extensions to ORM include on the one hand allowing quality metadata to be associated to context elements, which partially addresses one aspect, trust, related to data uncertainty, and on the other hand providing support for derived attributes, which in particular allows higher level context information to be automatically computed from lower level context information. They also add time constructs, but no space construct, thus failing to be able to describe the spatio-temporal features of context objects and associations that are so essential for LBS contexts. The same limitation, i.e. some time but no space, flaws the context model of the DAIDALOS European project[2], a generic conceptual data model very similar to the one by Henricksen and Indulska. The DAIDALOS model includes an activation/deactivation functionality, to express which context elements are active, associating it to attributes and associations only. Instead, support for multiple representation (another essential feature to enable LBS to handle context at various levels of granularity of from different perspectives) is planned as future work, while the same feature is already partly supported by Henricksen and Indulska via their concept of alternatives. Our approach also assumes that a powerful conceptual model (or an equivalent ontological formalism) is used to describe context data. We advocate that MADS, the conceptual data model developed by our laboratory, is best suited for the task as it fully supports spatio-temporal features as well as multiple representation.

Not surprisingly, the last trend in context modeling is to model context data as a context ontology. An example is the CONON context ontology [WGZP04] where context definition is separated into the definition of an upper ontology, i.e. a high-level ontology capturing general features of basic contextual entities (e.g. location, person, activity, service, device), and the definition of additional domain-specific ontologies. The advantage of using an ontology formalism, e.g. OWL-Lite, is its support of reasoning, a functionality that database technology and its conceptual models do not support yet. CONON uses reasoning for checking the consistency of context data and for inferring high-level context data from low-level context data. However, the authors do not explicitly state which consistency check they want to perform. As for inferring high-level context data, the same functionality may be supported by conceptual models allowing for derived attributes [HI04]. Thus, the benefit for CONON of using an ontology remains questionable.

For completeness of this short survey on context modeling it should be mentioned that multidimensional modeling techniques have also been investigated for dealing with context data. This is a computational (rather than modeling) trend focusing on using context data as raw data in a data warehousing perspective.

[1] ORM: http://www.orm.net/

[2] hrefhttp://www.ist-daidalos.org/http://www.ist-daidalos.org/

The goal is to use OLAP processes (in particular statistical and trend analysis) in order to derive higher-level context data from basic context data. Jensen et al. [JKPT04] develop a comprehensive solution for data warehousing with spatial data, focusing on dimensions based on partial containment. Context Cube [HLA+04] and [RSP05] apply the traditional cube concept to context data and show various application frameworks where higher-level context can be computed from a cube. One of the examples deals with inferring activity of seniors at home from the raw data collected via multiple sensors monitoring the movement of the persons within the house.

5.2.3 Context Classification and Management

Classification is a basic and very useful technique to shed some light onto a complex domain so that some generic understanding can emerge. To facilitate understanding what context data can be and how it can be handled, several proposals for classifying context information have been published. Basically, a distinction can be made between operational and semantic categorizations [vBFA05]. The operational classification in [HI04] is based on how context data is acquired and classifies context data as sensed, user supplied (static or profiled) and derived (i.e. higher level information). Each class may be characterized by e.g. a specific quality metrics. Semantic categories group context data according to the concept they are related to, irrespectively of how they are acquired. For example, Dey et al. [Dey01] introduce four context categories (identity, location, status or activity, and time) and apply them to places, people and things, thus determining twelve possible context classes. The characteristics of the classes driven from these two types of classification are discussed in [vBFA05]. As our work focuses on semantic issues in LBS, we naturally follow the semantic approach to determine the categories of context that will be discussed hereinafter.

More efforts, in particular from the software engineering community, have been devoted to the development of context management frameworks. A variety of tools/prototypes are available to handle acquisition, storing, aggregation, and delivery of context information in different abstractions for context consumers, thus facilitating the design and development of context-aware applications. Examples include Context Toolkit [DSA01], the platform-specific tools for the Symbian [KMK+03] and Solar [CK02] platforms, and IBM's Context services [LSI+02]. For example, [KMK+03] designed a client-transparent infrastructure supporting basic features of context such as being noisy, uncertain and rapidly changing. The infrastructure allows clients to subscribe, query and use any context information. The system is equipped with a Bayesian reasoner to supply and deliver context information ranging from atomic to higher-level information, and holds an ontology that defines the context structure and concepts to enhance context reuse and sharing.

5.2.4 Summary and Introduction to Our Approach

Most of existing research on modeling and using context information focused on either ad-hoc development of context-aware applications or the development of infrastructures providing generic context management services to their users. Fewer efforts have been devoted to define generic rules and structures enabling an intelligent use of context data for the selection and dissemination of information, i.e. to determine when, where, and what information can be disseminated to which users in broadcast solutions (push style), or on request, what information can be relevant to satisfy user's requests (in pull style) given the current context. In particular, existing conceptual data modeling approaches provide limited expressive capability for representing context data and their inter-dependencies. For instance, in [HIR02], the location context is just encoded as coordinates, but using this single data type seems inconvenient to directly support interactions with end-users. Instead, a coarser representation of the location, e.g. "at office" rather than the office coordinates (x,y,z), can be more convenient and more informative for user's dialogs with the LBS. Ontology-based approaches have a great innovative potential thanks to their solid logic-support, their easiness for reuse and extensibility, their capability to represent data in multiple representations and their suitability to serve in Semantic Web solutions, but there still exist a series of open questions, such as how to represent the spatial and temporal features of context entities especially when the context entity is moving and evolving, and how to describe the relationship between the context value and quality or effect of services. Finally, data warehousing approaches provide only limited support for specifying and reasoning on the rules defined by context-aware applications or end-users.

Our vision of and contribution to context data management is basically methodological and structural. The methodological perspective provides pragmatic answers to questions such as what is context (i.e. what context information is to be kept), how shall we structure context information and how it is used. Similar to the duality, advocated in CONON [WGZP04], between upper ontology and domain ontology, we assume a duality between an initial version of the LBS context data and subsequent enriched versions progressively built during the operation of the LBS. The initial version may be created from scratch by the LBS administrator, or be imported from some LBS context provider, or be automatically extracted by some knowledge extraction process embedded in the LBS and capable of learning from the available sources what are the relevant characteristics of the local context. This initial version of the LBS context (equivalent in purpose to CONON's upper ontology) contains context data describing the specific local environment, i.e. generic knowledge about the region covered by the LBS, independent from the services that will be made available to users once the LBS is operational. For example, local cultural habits typically provide such kind of initial context data. Further running versions of the LBS context result from augmenting the initial version with all context data driven from the definition of the service profiles that are progressively added into the LBS. For example, the context extraction process may examine the pre-conditions specified within a service description. Each pre-condition makes the service dependent on the current query or the current profile of the user or on the current state of the real world (the current "situation"). The latter is relevant to context maintenance and may trigger context enrichment. For example, if a service description has as pre-condition "open on weekdays only", it is inferable that context data must be able to determine if the current day is a weekday or not. If that information is not already in the context, it should be added to the context.

Context creation and enrichment, as well as later context use, is facilitated if context data is structured into intelligible semantic categories. The initial version of the LBS context contains predefined categories felt to be generically important to discriminate between different aspects of context. Space and time are in our opinion two categories that are inherently part of any initial LBS context. Environment, communication framework and socio-cultural features are additional examples. The enriched versions may add service specific categories, as they appear to be important to discriminate among the services made available via the LBS. For example, a religion context may be added if there are services that are religion-dependent, and an age-class context may be defined if data is available on how different services may target various age classes. Given the richness and complementarities of context data, the context repository holds many alternatives describing different context configurations. In terms of context usage, this entails the necessity to be able to identify which elements form the current/active context and to know what it means to change from the current context to another context.

From the structural perspective, we favor the adoption of a suitable conceptual model, namely MADS, for describing contextual data elements, their structure and their interrelationships. A MADS description can be translated into an equivalent relational schema and associated constraints and triggers. A system of predefined triggers allows the hosting relational DBMS to perform ontological reasoning on the MADS equivalent schema [AJPS07], thus providing a solution that combines advantages of both the conceptual and the ontological modeling approaches. Moreover, MADS multi-representation mechanism can be used to characterize the elements that belong to the current context, thus fulfilling another requirement for context management. This stated, the following sections in this chapter mainly abstract from the MADS background while presenting the basic constructs for context definition: classes, relations and constraints that we propose for handling context data. We rather focus on the relations and constraints that are typical of context data and of their use in matching service data (matching context with user profile data is discussed in the next chapter on user profiles).

We conclude this introduction by stressing again some essential targets in context modeling and management in LBS:

- **Spatio-temporal scope.** Context is heavily spatial and temporal dependent. Its spatial and temporal features need to be accurately described and processable. Therefore we discard as not suitable approaches where these aspects are not taken into account or are badly represented (for example, by representing location as an individual entity/class related to context entities that hold a spatial extent). Processability means that the chosen modeling approach has also to provide for spatial topological relationships and temporal synchronization relationships, so that at least basic reasoning on space and time can be supported. For example, the LBS may infer high-level context data on user's activity (i.e. determining current activity as ”shopping”) from low level data showing that the user position is currently topologically inside the extent of a shopping mall combined with the knowledge that the current timing is within a day qualified as ”weekend”.

  Knowledge of spatio-temporal contextual elements (as described by space-and-time varying attributes in MADS) is also essential when monitoring user behavior to detect context changes. For example, a visitor to a museum keeps moving (her/his position changes continuously) and the moves may trigger a (discrete) change of context, e.g. when moving from an exhibition room to the museum cafeteria the context changes from cultural to leisure or food, entailing different information requirements. LBS aiming at being always up-to-date in their management of current context have to be able to monitor spatio-temporal phenomena to automatically detect possible context changes.

- **Multiple representations.** Like factual information, contextual information may call for many alternative representations, which make its interpretation and use more complex. Context providers and context consumers may have different understandings of the same contextual information, and possibly in different resolutions (also called level of detail or level of abstraction). Hence, it is necessary for LBS designers to determine *which context in which resolution is relevant to a given task.* For instance, to look for a nearby restaurant, the current location in terms of coordinates is important, but to answer whether a low-cost airline office exists the precise user location is irrelevant, a coarser location value that just identifies the city in which the user is located is sufficient. Another example showing alternative representations of the same concept in different contexts is the definition of 'at leisure' in terms of timeframe as one of {*after daily work, on vacation, at weekend*} or in terms of activities as one of *doing sport, shopping, visiting museum.*

- **Customized context semantics.** In daily communication, humans (end-users and service providers) tend to use some predefined context terms that implicitly convey a possibly personal contextual semantics. The definition of customized contextual terms is often based on the observation and knowledge of local conventions or individual habits. For instance, 'at lunch-time' is used in Switzerland as denoting a temporal interval usually from noon to 2pm, and may implicitly suggest a casual atmosphere, i.e. a context in which information not related to work can be pro-actively pushed. By specifying their personal semantics of contextual terms, users can tune the LBS to adjust automatically to their view of the world. For instance, a Spanish user visiting Lausanne may query for a restaurant open at lunch-time, but his view assumes lunch-time to denote the 2pm-4pm time period. To get a personalized answer, the user must be able to explicitly state her/his definition of "lunch-time". Similarly,'after work' for a regular-hour worker denotes the interval from 18:00 to 24:00, but for a night-duty worker it means an interval from 8:00 to 17:00. Unfortunately, few approaches provide users and service providers with the facilities to express their own context semantics with ease.

- **Condition and dependency.** Contextual data can be used as hard or soft criteria in the selection of relevant services. Hard criteria lead to discard the services that do not meet the criteria, while soft criteria are used to order the set of selected services. For example, users moving on highways are normally interested in the services ahead of their current driving location rather than those behind. This determines a hard criterion to select services based on their location. In addition, the selected services can be sorted according to their distance from the user's current location (soft criterion). The service selection will also use current time (from the context data) to make sure that only currently available services are selected. Therefore, conditions and dependencies between contextual data and user/service data have to be carefully taken into account to elaborate the best possible answer.

5.3 A Framework of Context Interactions in LBS

Context information, like service information, is twofold. It includes the description of contextual data that has been selected as useful for the LBS and has to be maintained during LBS operation, and it includes the specification of contextual data that is specific to a given interaction with a user or specific to interactions with a given service. The first component, the generic local context, materializes the context module we have mentioned as being part of the core ontology (cf. chapter 3). It is described in detail in the next sections. The second part, a multiplicity of specific contexts, is stored in a context repository under the control of a context manager module within the LBS (cf. Figure 5.1). The context manager provides query and maintenance functionality through traditional management APIs (not shown in the figure). It interacts with the various sources of sensed context that capture and supply context data in the diverse resolutions and abstractions that are needed for the LBS. The term sensed context is used here loosely to denote any source for context information, be it a real sensor or a web page or any other information holder from which the LBS may obtain data to initialize and maintain the context ontology. The context manager also interacts with the user and the services worlds for acquisition of additional context data: context data extracted from interactions with the user and, similarly, context data extracted from interactions with service profile definitions. This architecture is illustrated in Figure 5.1 and conforms to a rather standard view of context management infrastructure (see, e.g. [LSI+02]). We do not develop a discussion of the architecture any further, as related issues are not a concern for our work focusing on semantic aspects. We directly proceed with the description of the generic local context embedded in the core ontology.

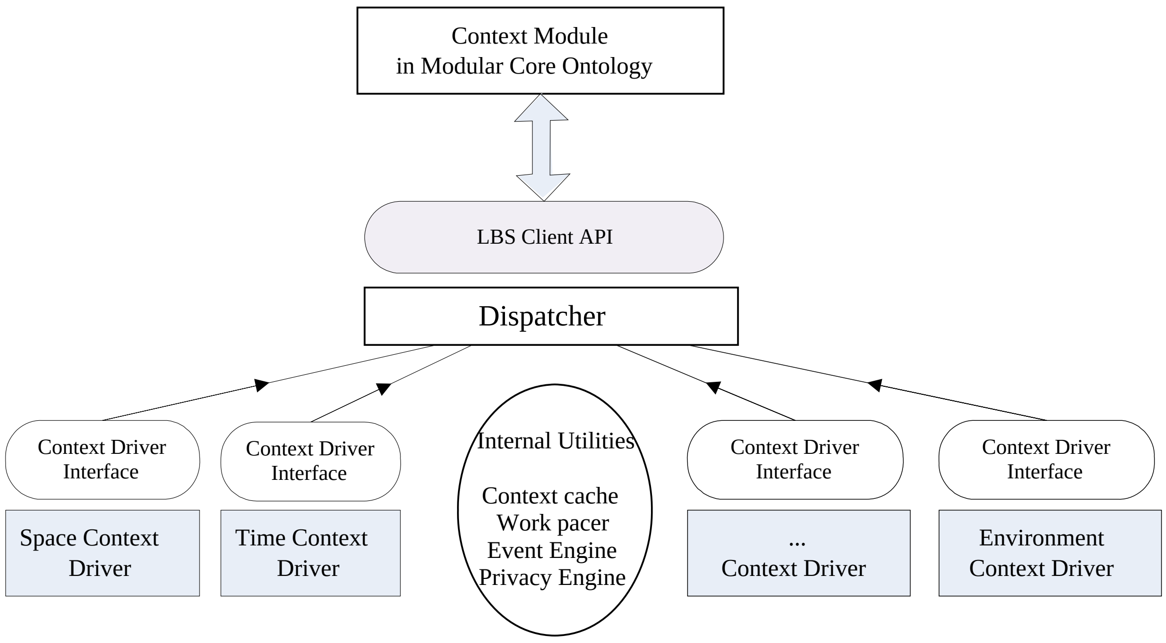


Figure 5.1: A general framework for context data acquisition and dissemination in LBS.

5.4 The Context Module in Our Approach

As discussed at the beginning of this chapter, context is a collection of information that may be useful in the selection of services by the LBS. For better comprehension and possibly more efficient management, context is structured as a taxonomic tree, e.g. the one illustrated in Figure 5.2. From a semantic viewpoint, each single element of context data can be seen as a (partial) context in itself. Hence it would make sense to organize context data as an is-a hierarchy of contexts, from the largest one (the all-embracing context) to the contexts with the narrowest scope. This view of context may be formally correct (i.e. consistent with the assertion that everything is context), but a more realistic or pragmatic (and more intuitive) understanding of the concept suggests that context is a multi-faceted complex concept. In this view the context tree is seen as a (de)composition structure, where an intermediate node is composed of its successor nodes. For example, referring to Figure 5.2, we can look at the generic top-level concept of context as a collection of many domain-specific contexts such as space, time, etc. However, the more intuitive view sees context as composed of a spatial component, a temporal component, etc. The duality of interpretations between composition and is-a is due to the fact that in case of contexts the nature of the components is the same as the nature of the composed. On the other hand, some of the edges in the tree are definitely is-a links, e.g. the arc between Calendar and LocalCalendar. Unwilling to enter a philosophical debate on whether a part can be the whole, we take the pragmatic approach and adopt the solution to model the tree as a composition tree that may occasionally include is-a links.

As we did in the previous chapter on service profiles, we use OWL-DL and SWRL to represent the context information and the relations between context, services and users. These languages have the expressiveness and simplicity that allow a concise and formal description of the main concepts, which is what we want to do here. Details about the properties of the concepts are left out. Therefore, each of the classes and relations that we discuss hereinafter should be seen as complemented with a full description of their properties, including the spatial, temporal and spatio-temporal properties, according to a mapping into OWL-DL of the MADS modeling constructs and rules [PSZ06].

Let us now comment on the context structure shown in Figure 5.2. First, the root of the tree is the most abstract concept of context and basically serves as input to any search for contextual data. In a non-modular view of the LBS ontology, it singles out context data from other types of data such as service profile data and user profile data. Using OWL-DL axioms this disjunction is stated as:

$\mathsf{Context} \subseteq \neg\ \mathsf{Service}$

$\mathsf{Context} \subseteq \neg\ \mathsf{User}$

The next level in the context tree shows the different semantic categories of context. This level is application-dependent, but as most LBS tend to be used for similar purposes some of the semantic categories can be considered as inherent to LBS functionality. This is certainly the case in our opinion for the space and time categories, which directly correspond to the concept of "location-based" services. But other categories are very likely to be generic and cover a large spectrum of the potential application-domains for

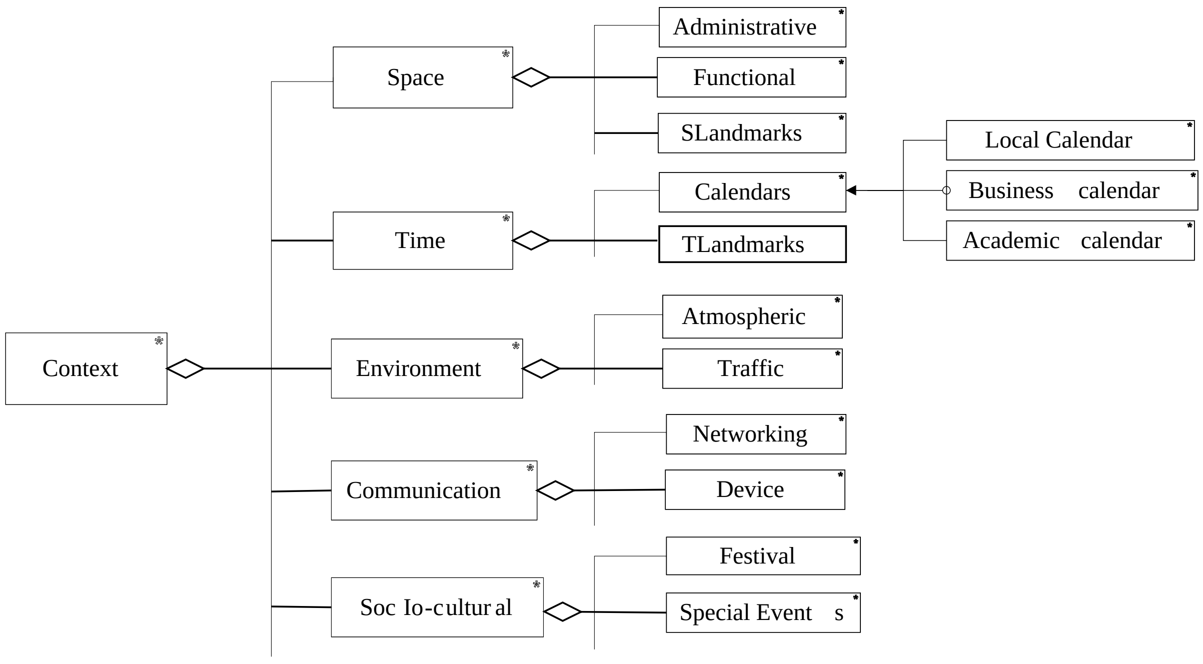


Figure 5.2: A possible taxonomy of context classes in LBS.

LBS. We include here as examples the categories that inform on local environmental conditions, available communication facilities and local socio-cultural habits and norms.

According to the taxonomy in Figure 5.2, the context classes and their taxonomy can be defined in OWL-DL axioms as follows:

$\mathsf{Communication\_Context} \subset \mathsf{Context}$

$\mathsf{Communication\_Context} \subseteq \neg\ \mathsf{Socio\text{-}Cultural\_Context}$

$\mathsf{Communication\_Context} \subseteq \neg\ \mathsf{Environment\_Context}$

$\mathsf{Communication\_Context} \subseteq \neg\ \mathsf{Space}$

$\mathsf{Communication\_Context} \subseteq \neg\ \mathsf{Time}$

$\mathsf{Network} \subset \mathsf{Communication\_Context}$

$\mathsf{Network} \subseteq \neg\ \mathsf{Device}$ etc.

The above axioms only show creation of context classes. These axioms have to be complemented with the definition of the properties for each context class, e.g. their spatial and temporal characteristics. As for the classes in the service module, a context class can be a primitive class, an enumerated class, a set-based class, or a discriminant class as defined in chapter 3. For example, the temporal context may support multiple calendars, including a high-school calendar defined as a discriminant class as follows:

$\mathsf{HighSchoolCalendar} \equiv \mathsf{Calendar} \sqcap \forall \mathsf{hasProfessionalDomain.HighSchool}$

The discriminating property hasProfessionalDomain informs on the domains where a calendar can be applied. If information in the user profile (e.g. the profession element) shows that the user is a high school student, and the user query calls for choosing a specific calendar (e.g. the query involves concepts such as "beginning of the year", "summer semester", "vacation period", concepts that the LBS ontology shows as having a calendar-dependent interpretation) the LBS, knowing from the context that different calendars depend on professional domain, can determine with its ontology that high school students belong to the high school professional domain and consequently choose `HighSchoolCalendar` as the calendar that corresponds to this domain and applies to the running query.

5.5 Context Classes

5.5.1 Spatial Context

This context category is meant to provide information about spatial features that help in understanding and reformulating the references to space (spatial predicates) that may appear in a user query and in service profiles. It includes subcategories that define the spatial extent covered by the LBS and its possible semantic interpretations.

GeoContext. The `GeoContext` class holds the basic definitions about the geographical features characterizing the current LBS. First and most obvious it holds the definition of the spatial extent covered by the local data available to users. This materializes what "local" means. For example, it may hold the spatial extent of the city of Lausanne, or of the Canton de Vaud. Assuming that the LBS uses a background GIS database to recover data about the area it services, the definition of the "local" extent will be added to retrieval queries to the GIS as an additional topological inclusion predicate ensuring that only elements within this area are returned by the GIS. Alternatively, if the local region is an object known by the background GIS, the value of its local extent can be left within the GIS and the queries reformulated by adding a predicate referring to the geometry of the local region object. Other properties in `GeoContext` may hold coordinates locating important singular points, such as the nearest/main airport, the capital/chef_lieu of the region, the main railway stations, etc. This information allows reformulating queries asking e.g. for a hotel near the airport, where it may safely be assumed that the user implicitly refers to the nearest/main airport. Generalizing the concept of singular point leads to the concept of geo-reference we introduced in chapter 3. The concept includes the concept of landmark, which denotes places, buildings, monuments, etc. whose name is frequently used by people as a spatial reference (e.g. in proximity expressions like "beyond the Eiffel Tower when coming from the Invalides"). `GeoContext` may include the local landmarks to enable the LBS to process queries and service descriptions referring to them. Data on landmarks can be directly acquired from a domain expert (e.g. the local tourism office), automatically extracted from web pages [TLKT01], or incrementally derived from service descriptions and user queries.

Administrative Spatial Context. It is known that the spatial extent of interest, defined in `GeoContext`, can be decomposed using a variety of criteria leading to different decompositions. One frequently used criterion relates to the administrative organization of geographical space. In general, this administrative view is expressed as a containment hierarchy. For example, in federal states such as USA or Germany the *country* consists of states, each *state* consists of *counties* and so on. Other administrative partitions exist due to different local history or cultural context. For example, in Switzerland, the *country* is composed of cantons and half-cantons, each full/half *canton* is composed of districts, and a *district* is composed of *communes* as shown in Figure 5.3. The containment hierarchy stored in `AdministrativeSpatialContext` holds the definition of the extents of each piece of administrative space and allows reformulating in geographical terms specifications given in administrative terms. For example, the specification that a pizza delivery service limits its deliveries to the district of Morges will be reformulated as limited to points within the spatial extent of this district. The reformulation allows computing whether the user requesting a pizza delivery can be addressed to this specific service. Other types of administrative spatial partition can exist in parallel to the previous one. For instance, Switzerland's split into three linguistic regions, according to their local official languages (German, French, Italian), may also be relevant. Here we just have a one-level split, no containment hierarchy. The two parallel taxonomies in Figure 5.3 can be easily defined by the OWL axioms rdfs:unionOf and owl:subclassOf. The axiom owl:oneOf can help to define the class *LinguisticRegion* by enumerating the component *district* individuals. Administrative classification data can be extracted from web sources, e.g Wikipedia, or provided by a domain expert.

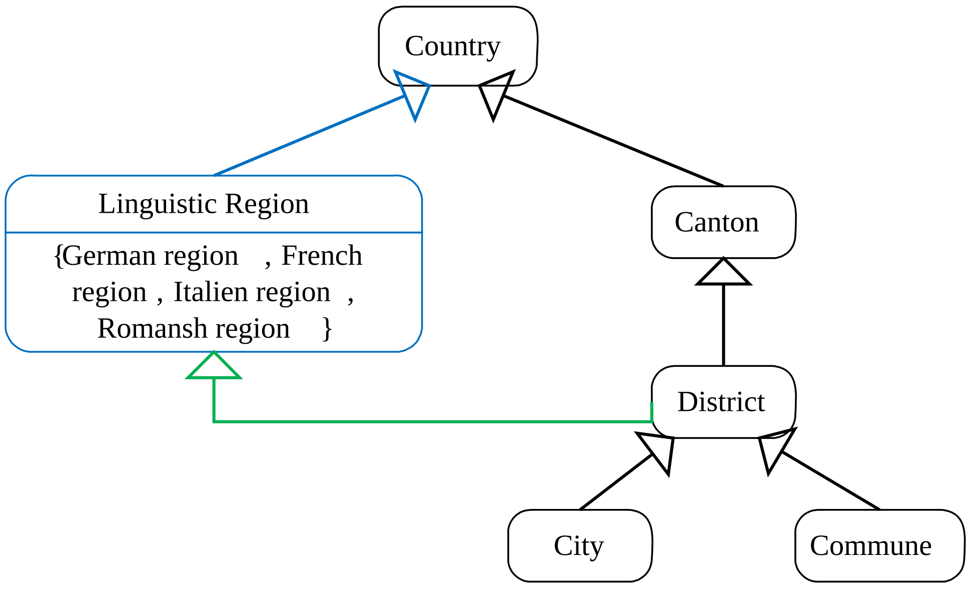

Country
Linguistic Region
{German region , French region , Italien region , Romansh region }
Canton
District
City
Commune

Figure 5.3: A hierarchy of Administrative Spatial Context classes in LBS.

Once again, instead of explicitly holding the values of the spatial extents of the different spatial regions, the context repository may just hold the identification of these regions and use their geometry stored in a GIS to express relevant spatial predicates.

Functional Spatial Context. Another frequent view of space is to single out spatial extents based on some property of the corresponding object. For example, based on the property "function" of venues such as buildings, shops, etc. (where function is understood as expressing the main activity supported by services located in the venue), an area can be characterized as a "shopping area", possibly including a "food court", while other areas are characterized as "entertainment area", "cultural area", "natural area", "business area", "campus", "technology park", "historical zone", etc. These concepts are frequently used by people on the move as well as by tourism offices, hence it seems adequate to have them materialized to form a dedicated decomposition of space. Differently from the administrative decompositions above, this "functional" decomposition does not cover the region of interest. It points instead at some specific areas within the region. The purpose of the context data in this respect may be to specify which kinds of functional space exist locally and where they are located. Such data is likely to be required from a domain expert. Its automatic computation is difficult to achieve due to the relative fuzziness of the concepts. For example, the class `ShoppingDistrict` can be defined as a subclass of `FunctionalSpace`.

5.5.2 Temporal Context

As for spatial aspects, temporal aspects have already been addressed in chapter 3. We provided the definition of the temporal data types and discussed the concept of temporal reference. Temporal references exist for various temporal granularities. At the hourly level, references of type `Instant` include *noon, midnight* whose meaning can be regarded as context-independent, as is the case for day granularity references such as *yesterday, tomorrow*. References of type `TimeInterval` designate concepts such as *morning, evening, lunch-time*. For day granularity, temporal references of type `Instant` are the time equivalent of spatial landmarks. They include singular days that people are used to refer to, e.g. *Christmas, Easter, New Year, Independence Day*. Temporal references of type `TimeInterval` include *spring, summer, autumn, winter, Christmas vacations, high-season, low-season, mid-season.*

Most of these temporal references are context-dependent. For example, which time interval is exactly meant by morning/afternoon/lunch-time depends on the cultural context, e.g. local habits or user habits. Some people would assume morning starts at 7am, others would make it start at 9am. Christmas and Easter are strongly related to Christian frameworks, they do not exist in a Muslim framework. New Year is celebrated at different times in different countries, and so is Independence Day because the various countries have their own independence day. It is therefore important that context classes provide the values that are appropriate for a given LBS, its service description and the queries it may get.

Other contextual temporal constructs include constructs such as *Calendar*. In the western world the Gregorian Calendar is the implicit temporal framework to talk about dates, but for other countries, cultures and religions different calendars may apply. Local calendars may still be kept as a traditional way of referring to time. For instance, the Chinese Lunar Calendar can provide local indications on seasonal changes. Other calendars, in some way different from the generic calendars above, may be widely used for specific and local needs. For example, calendars may hold for a specific region, such as a calendar for a *Canton* or a *Linguistic region* which may show official local holydays that only exist within the region or have a timing that is specific for the region. For example, *Mothers' day* is on a different date in France and in Switzerland. Calendars may also be specific to a professional framework. For example, events relevant for students and teachers may be related to an academic calendar that has its own definitions for the two temporally disjoint classes *university vacation* and *university semester*. Calendar subclasses can be further given, e.g. *university vacation* can be defined as *Easter-holiday vacation, Summer vacation, New-Year vacation and Winter vacation*. The definition of these academic periods may be relevant beyond the academic world. For example, the public transports in Lausanne have different timetables for *university semester* and *university vacation*. Finally, it is worth noting that the temporal class *Summer vacation* may correspond to different values for different perceptions, e.g. summer vacations seen by university users is different from summer vacations seen by high-school users. Instead of defining two academic sub-contexts to differentiate between university and high-school, it is possible to keep summer vacations as a single item with two representations (i.e. as a perception-varying attribute in MADS terms). This is one of the cases for multi-representation. In our context definition, the OWL property hasPerception is defined as *necessary & sufficient conditions* to declare its value of mandatory property hasInterval depending on the perception. Regarding how to apply it in service matching and in articulating user profiles will be discussed in Chapter 8.

5.5.3 Spatial and Temporal Variability

Much of context data is strongly dependent on space and time. For example, which landmarks exist depend on what is the covered region. In MADS terms, this is a space-varying information and can be described as a space-varying attribute, which means that its value is a function whose domain is a spatial extent and whose range is the value domain for the attribute (e.g. string, integer). Similarly, much information is time-varying, i.e. it changes as time passes. Its evolution can be kept in a time-varying attribute, i.e. a function from time to a data value domain. Combining the two, information can be both time and space varying (e.g. the set of landmarks), we call it spatio-temporal information.

As we mentioned for multi-representation, associating space and time variability to context data is also a way to limit the complexity of the description of context data. For example, let us assume that the SpecialEvent context class has an entry for fireworks used to celebrate a national festival, and there are many different fireworks in different places within the city at different times. Instead of defining many contexts specific to a place and a time interval, one can simply define a single set-valued attribute fireworks equipped with a standard tabular structure that adds ”where, when” data to each firework. As we have seen in Chapter 3, in our OWL-based context structure, the normal (i.e. static) spatial and temporal characteristics of context are encoded as properties whose range is one of the spatial and temporal data types discussed in Chapter 3. Spatio-temporal characteristics are defined using set-valued properties with ”where, when” components, such that the values of where and when are constrained to ensure that where is within the region of interest and when is a time-interval with hour granularity included in the interval [16:00, 24:00] (assuming local habits plan for fireworks only in this time period).

5.5.4 Environment context

The term *environment* is used in the sense given by the Oxford dictionary: ”physical surroundings and conditions, especially as affecting people’s lives”. Most frequently quoted examples of such environmental conditions for LBS include atmospheric conditions (weather may have an influence on people’s choice of activities) and traffic conditions (as people may adjust travel plans to avoid potential traffic delays). In more localized context-aware applications, e.g. ambient intelligence, environment context focuses on the interactions between the user and her/his immediate surroundings, e.g. using sensor data for automatically adjusting the room light and TV sound to user's activity.

Environment data may be spatio-temporal, i.e. both space and time varying. Atmospheric and traffic data are typical example. Querying such data implies the specification of a location in space and an instant in time in order to get the right answer (e.g. asking for traffic conditions the coming weekend on the highway to a ski resort). The location in space is easily derivable as being either the current user location or the targeted user location, i.e. the location where the user plans to move to do something or to get the service that (s)he is requesting. The relevant instant in time usually is either the current time or some specified time in the future. This points to the fact that, contrarily to what is provided in most approaches to temporal databases (i.e. the functionality to handle past and present data), temporal management of context data calls for functionality primarily to handle future data in addition to current data. Past data is of little concern except for any machine learning mechanism that would be embedded within the LBS. Future data management can be enhanced by adding quality information to predictions, e.g. accuracy, uncertainty and timeliness based on some quality metrics. A fixed format for predicted information can be specified by the LBS (e.g. "when the prediction is formulated, for when and where the prediction holds, what the prediction says, the likeliness of the prediction") and implemented as e.g. a prediction data type. A prediction for a precipitation item could be: "`predictionDate`= "2007-03-01, 18:00", `targetedDate`="2007-03-02", `targetedPlace`="Lausanne", `precipitation`="Snow", likeliness=0.9"

Intelligent use of environment data requires knowledge about which data is useful for which purpose. For example, assume a user is willing to reach a ski resort from the city (s)he is in at the moment. The LBS should know that, given a query on traveling between locations A and B, answers depend on means of transportation. In particular, environment context data on traffic density between A and B is relevant only if the travel is by car, not if the travel is by train. The knowledge on which context data to use and how comes from the association between services and context on the one hand, and on the other hand from the association between context and query (for facts that depend on specificities of the current query) or between context and user profiles (for facts that are user-dependent but not query-dependent). This will be shown in the chapter on query processing.

5.5.5 Socio-cultural Context

The environment context depicts the physical status of the local region. The *socio-cultural context* depicts in some sense the mental status of the local region. Mental status refers to the specific local understanding of concepts, facts and rules that may be involved in servicing users of LBS. Some of these specificities have already being identified and included in the spatial context (e.g. what are local landmarks) and in the temporal context (e.g. what are local holydays), because of their primarily spatial or temporal nature. All other non-physical local characteristics are candidate for inclusion in the socio-cultural context. For example, whatever is a "typical" feature could be recorded here if of interest to users or to service description. "Typical" features include typical local souvenirs (e.g. tourists to Geneva tend to buy watches, clocks, chocolate), typical local food (e.g. fondue and white wine for the Swiss Romande, minced veal and rösti for Zürich), special local attractions (e.g. the Olympic museum, the Collection de l'Art Brut and the Ballet Bejart for Lausanne). Local socio-cultural specificities include habits and social norms, such as meeting time is to be understood as a sharp specification (i.e. for a meeting at 2pm people arrive 5-10 minutes before 2pm) in Switzerland and as a loose specification elsewhere (i.e. a meeting at 2pm will actually not start before 2:15pm in France and start any time after 2:30 in Italy). This information will be used whenever the LBS user asks when (s)he has to leave the hotel to attend a meeting.

One socio-cultural aspect that frequently influences user's search for services is user's activity. Knowing what the user is doing (e.g. is the user on a business trip or on a leisure trip) may be important for example to suggest a hotel (e.g. close to the meeting place or close to tourist attractions) or a restaurant (more formal or more casual). Knowing that the user is shopping may prompt services for buying assistance (providing suggestions and comparisons). Knowing what the user plans to do may be used to elaborate schedules on request. The number of activities that one can think of, at different levels of detail, is practically unbounded. Examples include being at work, studying, relaxing at home, being on the move, attending a conference, and watching a movie. The LBS has to determine which activities are relevant in choosing a service for a given user query, and store the list of selected activities as part of context information. We propose to have an `ActivityContext` class as a component of the `EnvironmentContext` class. Properties attached to the activity context class may include the name of the activity, its category (e.g. professional, leisure, sport), whether the activity is an individual or a group activity, for group activities the different roles in the group (e.g. leader, participant, assistant), if any, and the constraints, if any, on the number of participants, whether it requires specific conditions such as daylight or a given season or given atmospheric conditions, whether it is indoor or outdoor or both, what is its minimal, average and maximal duration if relevant, whether it implies moving from one place to another, etc.

A major concern for activity management in LBS is how to acquire the knowledge about the activities of the querying user. Without this knowledge all activity-related data becomes useless. Obviously, the easiest way to activity acquisition is to ask for explicit input from the user, i.e. kind of asking what are you doing or planning to do. The alternative is to try inferring the activity from user interactions or user movement. The former (i.e. understanding what the user is doing from the questions (s)he asks) is difficult even for humans; hence it seems unlikely that such inference can be fruitfully implemented in a LBS. The latter can provide some insight and allow switching from spatial knowledge to social knowledge. First, human movement can be captured, for example using RFID tags or thanks to a GPS device, most likely as a discrete sequence of "when, where" data pairs. This raw data can be aggregated and interpolated to form a path. Analyzing the characteristics of the path, in particular the stops it includes (i.e. a point in the path such that the user remains in its close vicinity for a while), the LBS can turn the path into a set of trajectories taking the user from location A to location B. The LBS may also guess (from e.g. velocity) additional information such as the transport means, i.e. whether the user is walking, in a bus, in a car, etc. Matching user trajectories with background knowledge about the region the LBS may be able to infer the user activity. For example, if the user moves from the location of her/his hotel to the location of an office building (and stops there for a while) it is possible to infer that the user has gone to work or joined a meeting. If the targeted location is a conference facility, a reasonable guess is that the user is attending a conference. If, instead, the user stops in a department store, or walks slowly along a street in a shopping district, it is likely that the user is shopping. Further inferences are possible. For example, if the user stops for some time in an office building, it would be possible for the LBS to look for the companies located in the building to see if, given the professional profile of the user, is it likely that (s)he joined one of these companies. With more data the LBS could determine whether the user is visiting one of her/his customers or, conversely, visiting a supplier or service provider. Hopefully, this kind of inference will be sought only for special applications involved in security issues. Less sensitive inferences may try for example to determine which kinds of products a shopping user is interested in, possibly to suggest other shops to visit. Movement analysis may just be used for deriving a direction, to be used for example to constrain search for services for a user moving on a highway to preferably services that are ahead of the user's position. This short discussion shows that there is room for a `MovementContext` class as another component of the `EnvironmentContext` class. Movement context data would include the inference rule to extract information from analysis of physical movement, in particular to extract activity information. This rules would then by applied to a specific user and the resulting inferences stored in the user-specific context data that bridges between the user layer and the context layer. A model for trajectory description can be used to support knowledge about movement. Such a trajectory model is currently developed within the activities of our laboratory [SPD+08]. We discussed movement based on a physical path determined by RFID, GPS and similar devices. Movement can also be directly captured at a semantic level, e.g. as a trajectory from hotel to office, then to restaurant, etc., using input from the user's agenda, if available. Again, there are many possible inferences for semantic-based trajectories. We leave to the reader to imagine potentially useful scenarios. As a final remark, it is worth noting that not every activity can be inferred from movement. Assume a given user moving from a meeting place to a restaurant. It is not possible to infer from the spatial path, nor from previous and following activities, whether the walk for lunch is nothing but a utilitarian move, is used to continue the discussions from the meeting, or is used to socialize.

Other potential components of the socio-cultural context class are classes providing knowledge about local events and festivals (the kind of information found in What's On booklets). This kind of information is normally available from data providers such as local tourist officers and local newspapers and can therefore be directly extracted from the corresponding services. However, the LBS may be designed to keep some of this information in its context data and use its potential relationships to other data. For example, knowing that today there is the annual Marathon race in Lausanne, the LBS will be able to avoid suggesting a transportation or an activity that would be blocked because of traffic limitations implied by the Marathon. Notice that traffic restrictions could also be available from a provider of traffic forecast or monitoring (e.g. local radios have this kind of information).

5.5.6 Communication Context

In contrast with the previous one, this is a relatively low-level context component. It holds data relevant to the wireless communication between users and the LBS, including the *network*, *mobile device*, *channel* and the *communication* between the network and the device.

Contextual data about the *network* may describe its type (e.g. Bluetooth, GPRS etc.), signal coverage (e.g. spatial coverage of the network), the quality of the communication, the accessibility of the network (e.g. public/private or free/paid), the provider of the network, etc. This is useful in case there are services whose accessibility is network-dependent.

The *mobile device* and *channel* contexts hold the basic information about the user's mobile device. This may include the producer, mark, and type of the device, its identifiable network-type, its capability to support audio/video transmission, the capability of its software support, the signal quality in different geographical regions, etc. These data allow personalizing the physical communication between the LBS and the user.

Regarding the *communication* service, its context data may hold specific conditions or perceptions related to the communication, such as the network connection, upload/download speed, representation format, or user's identification and validation, etc. The communication context may be specialized for a specific user, for example to keep the preferred network or channel according to different social contexts. Users can configure their devices in different social context, for example switching off during a meeting, allowing/disallowing pushed information while in a shopping-center, or choosing a specific format for information presentation.

5.6 Context Relationships

The previous section has exemplified some context classes that we consider as typical in an LBS framework. More classes can be added depending on local characteristics. The key criterion for determining the classes to be used is their relevance in terms of capability to lead to more focused and more personalized service selection. This relevance is made explicit using dedicated inter-module links within the LBS core ontology. The role of these links is to connect context elements to the services whose availability or behavior is influenced by these elements, and to the user profile elements that can be identified as being context-dependent. Before we discuss inter-module links, we devote this section to a short discussion of relationships within the context module, namely a more detailed discussion of the composition (part of) relationship. The composition relationship has been presented in chapter 3 as part of the relationships that are generically used to structure the content of an ontology. In chapter 3 we focused on the is-a relationships. As stated in the beginning of this chapter, for the context module composition relationships, rather than is-a relationships, play the preeminent role. It is therefore worth defining finer modeling features for composition in the context module (yet this finer features can obviously be used within any module).

5.6.1 From Composition to Synchronous Composition

The *composition* relationship defined in chapter 3 simply relates a composed object to one of its component objects. At such a generic level we abstained from discussing how the assembling of the components into the composed can be ruled. Ruling composition means associating some constraints to the composition construct. To do that smoothly we first propose a definition for the composition construct, adding the potential for constraints to the definition of composition relationship in chapter 3. This results in the following definition.

DEFINITION 5.1. Composition Construct. *A Composition Construct, denoted as $\Subset(C_W,\ C,\ \mathcal{C})$, is a set of composition relationships between a composite class C_W and each of the classes in its composition cluster, i.e. a set of component classes, $C = (c_1,\ \ldots,\ c_n),(n \geq 1)$; and $\mathcal{C}$ is the constraint associated with C and C_W.*

If constraint $\mathcal{C}$ is empty, the construct simply gathers the component object, its composition cluster and the composition relationships in between, without adding anything to the composition relationships. This corresponds to the case where the cardinality constraints attached to the composition relationships suffice to express the desired constraints on the composition construct. We introduce two cases where $\mathcal{C}$ is not empty and holds a constraint on the spatio-temporal features of the involved objects. The first case defines $\mathcal{C}$ as a spatio-temporal constraint calling for spatio-temporal synchronization. The second case calls instead for handling the composition construct as a sequence of assembly. We denote the two cases as *synchronous composition* and *sequential composition*, respectively. The former corresponds in our opinion to the most intuitive and most frequent rule about composition, i.e. that the composed object results from a simultaneous assembly of the component objects. In other words, selected instances of the component objects are put together in some place at some given time and this produces an instance of the composed object.

DEFINITION 5.2. Synchronous Composition Construct. *A Synchronous Composition Construct, denoted as $\Subset_{SYN}(C_W,\ C,\ \mathcal{C}_{SYN})$, is a composition construct $\Subset(C_W,\ C,\ \mathcal{C})$ such that*

- *Let p^T and p^S be separately temporal and spatial properties of C_W. Let cr_i denote an instance of the construct, i.e. the set of instances of the composition relationships involving c_{wi}, the i^{th} instance of C_W. $\forall\ cr_i \in \Subset(C_W,\ C,\ \mathcal{C}_{SYN})$, $p^T(c_{wi}) = d^T$ and $p^S(c_{wi}) = d^S$;*
- *$\forall i \leq n$, p_i^T and p_i^S are respectively temporal and spatial properties of class c_i, with values d_i^T and d_i^S;*
- *$\mathcal{C}_{SYN}$ is described as an axiom: $\forall\ cr_i \in \Subset_{SYN}$, $c_{wi} \in cr_i$, $d_i^T = d^T$ and $d_i^S = d^S$.*

This case applies to spatial and temporal objects. It requires that the linked objects share a common spatial location and temporal framework for the specified properties. Only the spatial or only the temporal property may be specified,

In the example Figure 5.4, the context class *Weather* is composed of a set of component context classes such as *Temperature, AirPressure, Wind, Humidity, Cloudiness, and Precipitation.* Each of the component context classes represents one aspect of the composite context class *Weather.* All classes represent space and time varying phenomena. One instantiation of *Weather* makes sense only if all component instantiations of Temperature, etc. correspond to the same location and temporal snapshot. The synchronization requirement holds.

Example 5.1. Weather vs. Temperature, Precipitation, and Wind.

The axiom below illustrates how a synchronous composition construct may be defined between the Weather composite class and three component context classes, Temperature, Precipitation, and Wind. It shows the explicit declaration of the equivalency constraint on spatio-temporal characteristics, which characterizes the synchronous composition construct.

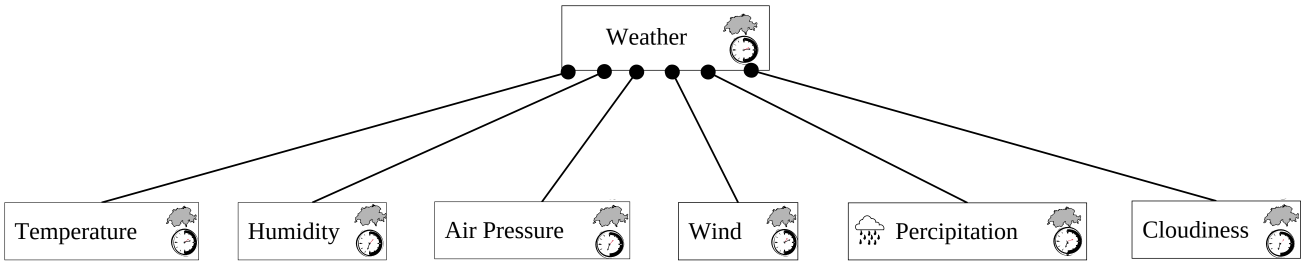


Figure 5.4: A synchronous composition construct.

$(\Subset_{Syn}(\text{Weather}, (\text{Temperature}, \text{Precipitation}, \text{Wind})) \Rightarrow$

exists (isUnionOf (Weather, Temperature, Precipitation, Wind)

and hasSpatial(Weather, $?\mathsf{s}_{Weather}$) and hasTemporal(Weather, $?\mathsf{t}_{weather}$)

and hasSpatial(Temperature, $?\mathsf{s}_{temp}$) and hasTemporal(Temperature, $?\mathsf{t}_{temp}$)

and hasSpatial(Precipitation, $?\mathsf{s}_{precip}$) and hasTemporal(Precipitation, $?\mathsf{t}_{precip}$)

and hasSpatial(Wind, $?\mathsf{s}_{wind}$) and hasTemporal(Wind, $?\mathsf{t}_{wind}$)

and EQUAL($?\mathsf{s}_{weather}, ?\mathsf{s}_{temp}, ?\mathsf{s}_{precip}, ?\mathsf{s}_{wind}$)

and EQUAL($?\mathsf{t}_{weather}, ?\mathsf{t}_{temp}, ?\mathsf{t}_{precip}, ?\mathsf{t}_{wind}$))

5.6.2 Sequential Composition Construct

More complex constraints in composition constructs are possible, including any sort of inequalities between spatial and temporal properties. In this section we point at another specific case, where the assembly of the composed object consists in a temporal sequence of acquisitions of single components. For example, an activity context class may describe sight-seeing tour activities, composed of several sub-activities, with a temporal constraint that organizes the sub-activities into a temporal sequence such that each sub-activity but the first one starts when the previous activity ends. In other words, the temporal interval associated to the composed activity is partitioned into a sequence of intervals associated to the sub-activities. We call this case a *Sequential Composition Construct.*

DEFINITION 5.3. Sequential Composition Construct. *A Sequential Composition Construct, denoted* $\Subset_{SEQ}(C_W, C, \mathcal{C}_{SEQ})$ *is a composition construct* $\Subset(C_W, C, \mathcal{C}_{SEQ})$ *such that*

- p^T *is a temporal property of* C_W. $\forall\, r_i \in \Subset(C_W, C, \mathcal{C}_{SEQ}), c_i \in r_i, p^T(c_i) = d^T$ *and the data-type of* d^T *is* *Interval;*
- p^T_j *is a temporal property of class* C_j *in* C. $\forall$ *component individual* $c_m \in C, p^T(c_m) = d^T_m$*;*
- $\forall\, c_{m-1}, c_m \in C$, *Meets*$(d^T_{m-1}, d^T_m)$*;*
- $\forall\, c_m \in C$, *Within*(d^T_m, d^T).

The Definition 5.3 relies on the two synchronization operators Within and Meets defined in the temporal module. They concur in specifying the constraint between the component contexts and the constraint between each component and the composite. Informally, temporal Within(t1, t2) means the instant/interval t1 is *during* interval t2 according to [All83]; temporal Meets(t1, t2) means the end of t1 *meets* with the beginning of t2 if both t1 and t2 are intervals. In some circumstances, the temporal operator Meets can be replaced by Before which has less strict constraints on t1 and t2. More details on how to encode them into the temporal domains in description logics can be found in [Sot06] page 26-29.

The above definition does not consider possible spatial constraints. Thus, *sequential* in Definition 5.3 just refers to *chronological sequence*. However, in reality, both spatial and temporal constraints are involved in certain sequential compositions. Frequently, the spatial constraints are dependent on temporal constraints making up for a spatio-temporal constraint. In the Example 5.2, we show how both the spatial and temporal constraints have effects on the sequential composition construct.

Example 5.2. Sightseeing tour vs. Sequential activities.

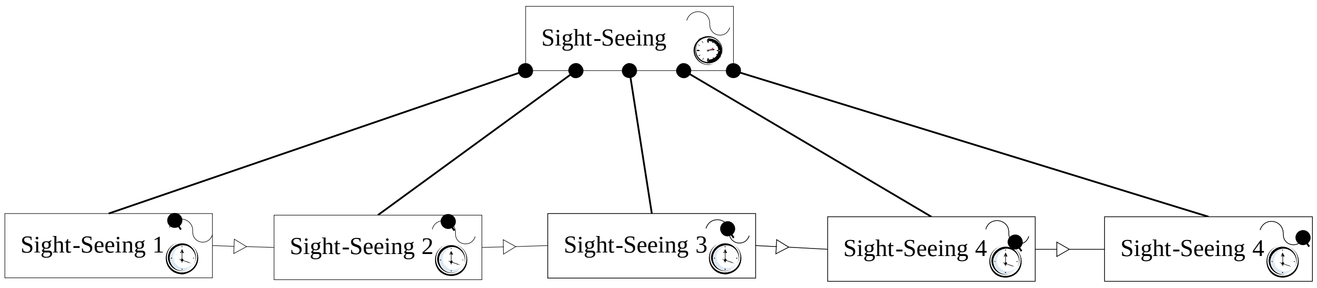


Figure 5.5: A sequential composition construct.

Figure 5.5 illustrates a sequential composition construct involving the sight-seeing activity as composite class, composed of five sequential sub-activities. The *sight-seeing tour* is regarded as a whole activity and spatially regarded as a trajectory, and it is composed of a set of attraction sites along the trajectory. For each site, the tour spends a certain time interval, and all of them are organized in a specific sequence. To illustrate the constraints, we give an example as follows: the whole activity is s = 'Afternoon tour near Ouchy', and component activities are s1 = 'visit Ouchy Port', s2 = 'Visit Olympic Museum' and s3 = 'Coffee-break near Ouchy'.

$\Rightarrow$ ($\Subset_{SEQ}(C_{Sightseeing}$, (C_{visit}, C_{break})),

exists (isUnionOf($C_{Sightseeing}$, (C_{visit}, C_{break}))

and hasIndividual($C_{Sightseeing}$, ?c_s) and hasIndividual(C_{Visit}, ?c_1)

and hasIndividual(C_{Visit}, ?c_2) and hasIndividual(C_{Break}, ?c_3)

and hasSequence(?c_i, ?i)

and hasSpatial(?c_s,?s_s) and hasTemporal(?c_s, ?t_s)

and hasSpatial(?c_1, ?s_{c1}) and hasTemporal(?c_1, ?t_{c1})

and hasSpatial(?c_2, ?s_{c2}) and hasTemporal(?c_2, ?t_{c2})

and hasSpatial(?c_3, ?s_{c3}) and hasTemporal(?c_3, ?t_{c3})

and Meets(?s_{c1}, ?s_{c2}) and Meets(?s_{c2}, ?s_{c3})

and Meets(?t_{c1}, ?t_{c2}) and Meets(?t_{c2}, ?t_{c3})

and Within(?s_{c1}, ?s_s) and Within(?s_{c2}, ?s_s) and Within(?s_{c3}, ?s_s)

and Within(?t_{c1}, ?t_s) and Within(?t_{c2}, ?t_s) and Within(?t_{c3}, ?t_s)

)

In the above axiom, we introduce two spatial operators Within and Meets. The former one Within(s1, s2) stands for the trajectory/region s1 is *a non-tangential proper part (NTPP)* of trajectory/region s2, i.e. NTPP(s1, s2) according to topological *RCC-8* relationships [RCC92]; the latter Meets(s1, s2) stands for trajectory/region s1 is *externally connected to* trajectory/region s2.

5.7 Multiple Representations in the Context Module

Multiple-representation is an essential character of real world phenomena, so it is for context data. Similar to [PSZ06], concern for multiple representation of context data mainly applies on the following issues:

- How context information is organized (in terms of data structure). For instance, the *connectionSpeed* can be a property, with value range {good, normal, bad}. Alternatively, it can be described by a set of sub-properties, such as hasUploadSpeed, hasDownloadSpeed, etc.

- How context information is encoded (e.g. in terms of dimension and unit). For instance, the *hasAntiVirus* property can correspond to value "yes/no", or more precisely hold the anti-virus software name. Spatial contexts can be represented in diverse resolutions and scales.

- How the context information in diverse representations is associated with diverse services. For instance, when the user is visiting a museum, her precise position, i.e. coordinates, is needed for providing the appropriate audio description of the art piece she is looking at. But, when the user finishes the visit and inquires for the bus-stops near the museum, a coarser location of the user(e.g. in the museum) may be sufficient for answering the query.

In general, a possible implementation of multi-representation is using a class *Representation* and a property *hasRepresentation* to denote the multiple representations feature of context data. Furthermore, the latter can have sub-properties, such as *hasScale* for *location* class, *hasUnit* (e.g. day or hour) for temporal class. To describe that a property of context is multi-represented, the representation information needs to be encoded in the property by declaring it as a representation-dependent property. The definition way is similar to the definition of space-dependent property.

When expressing the relationships between contexts or between the context and a service, it is necessary to add the representation information to the context class or context property, in order to make clear what representation of context information will be involved in the relationships.

5.8 Chapter Summary

Context-awareness enables LBS to adapt contextual changes in selecting services to better suit the user's needs. In literature, many efforts have been made in promoting the context-awareness of mobile services, ranging from designing architectures to collect and disseminate context services, to modeling context data with different approaches. In this chapter, we did not discuss in detail about the deployment of context-awareness in LBS, and assume that a set of context profiles can offer up-to-date context data to LBS through the dispatcher and client APIs and these context profiles can well communicate with context module in the core ontology. Rather, we concentrate on defining most important contexts to improve LBS's personalization and context-awareness rather than the contextual universe of world. Five main types of contexts are identified in LBS, i.e. space, time, environment, communication and socio-cultural contexts. For each type of context, we discuss their characters and functionalities. Considering the wide use of common sense knowledge in LBS, we introduce temporal landmarks and spatial landmarks to annotate locally used temporal/spatial terms in order to facilitate the user's query formulation. Due to the specific spatio-temporal constraints on composition relation, we provide two constructs *Sequential/Synchronized Composition Construct* in the context module. By employing the similar approach in MADS to model the multi-representation of context, a class Representation and a property hasRepresentation can denote the multi-representation character of a context class/property. In the next chapter, we will continue to discuss how to express relations between context and user, and between context and service.

Chapter 6

User Information Management

6.1 Introduction and Motivation

Personalization is a rapidly developing discipline. Its applications range from conventional information search, information sharing, and content representation, to customizing the interactions between users and mobile services. In the mobile computing environment, personalization technologies have to adapt to the essential mobile and dynamic needs of services and users. In our LBS setting, speaking about personalized services we stress that not only we want the LBS to be able to elaborate different answers to the same query depending on which user is querying the LBS. We also expect the LBS to adjust to the current ”role”, so to speak, that the querying user is playing while issuing this specific query. By role we mean the user may currently behave as an employee on the move for professional reasons, as a tourist doing sightseeing, as a sports person willing to exercise, as a hobby fan, etc. In other words, we want to associate not just one profile to the user, but as many profiles as the roles a user may play. Thus, different subset of user preferences can involve in diverse situation/role-based profiles, e.g. *conference membership* can be included in *professional profile* other than *tourist profile*. Moreover, it is quite common for a user to hold diverse preferences in different contexts [YAJS05]. For instance, in sunny summer the favorite sports are hiking and surfing, but if it rains the favorite one is indoor-swimming instead. Obviously, there is a dependency between the favorite sports (user profiling information) and the weather (context information). The context *Weather* acts as discriminating criteria potentially determining user's preferences. Therefore, on one hand, LBS must hold the knowledge on context concepts that are used in user profiles, e.g. *what does CollegeCalendar mean in a user profile*; on the other hand, LBS should be able to communicate with certain context repository to obtain the corresponding value of the context, such as *if (s)he is currently during the summer holidays according to this calendar*, and then encode the context-dependent profiles in query processing.

In this chapter, we firstly review the approaches of preference modeling and representation in literature, and then present the basic infrastructure of user-relevant information's interactions between LBS and user profiles. Secondly, we explain how LBS deal with the user-relevant information, i.e. to encode it in terms of concepts understandable by core ontology and further to employ them in query processing. Finally, we present our approach of defining and organizing the user profiles, i.e. defining some basic constructs to describe the classes, relations, properties, and dependencies/constraints and rules in user profile(s).

6.2 Related Work on Preference Modeling

As addressed in Chapter 2, user profiles functioned as an effective means in personalizing web-based applications, in terms of information filtering, content representation, information sharing, and privacy control. Definitely, these functions are the major concern when we manipulate user profiles and build up the knowledge relevant to users in LBS. In literature, user profile's content and representation vary from one application to another one, be they employed in static or dynamic computing environments. In this section, we review the prevailing approaches of preference modeling and representation and point out the major challenges in modeling and manipulating user profiles in LBS framework.

Most web-users have experienced how to define and apply their own user profiles in earlier web-information systems, where user profiles were simply represented as a set of categories in terms of keywords. This solution is popular in information subscription services, and usually associated with the temporal constraint, e.g. daily news subscription, monthly product catalogue newsletter. It mainly adopts the keywords-matching strategy in filtering information to the given user. It can assist users to customize the information delivery, but its poor expressive power and reusability result in failure to support the basic functionality in LBS, e.g. being context-sensitive and multi-domains. The other simple but popular approach is to formulate user profiles in a form format, i.e. composed of a set of conditions specified within a form. Richer than the category-based user profiles, it allows users to further customize the information selection by inputting some values and keywords for certain properties, i.e. expressed in terms of attribute-value pairs, e.g. Monster for job seeking. However, in this approach, user profiles are closely coupled with a given application, so that it is also lacking portability and unable to support evolution. Additionally, each user only has a single user profile, i.e. context-insensitive.

It is doubtless that a database approach can provide a solid logic basis and better expressiveness from the data management and modeling perspective. Its well-defined query languages can ease the query formulation by combining user preferences into conditions. In literature, [Kie02] and [Cho03] consequently proposed very similar frameworks to employ the database to model the user preferences. They both extended the relational model with formal first-order logics to express the preferences, such as $\succ$ is a preference relation over a schema relation R and $t_1 \succ t_2$ means tuple t_1 dominates tuple t_2 in $\succ$. For instance, assuming a schema relation R = (ISBN, Vendor, Price), a preference relation $\succ$ may be defined to state a preference for books available for a cheaper price, as follows: $(i,\ v,\ p) \succ (i',\ v',\ p') \equiv i = i' \wedge p < p'$. Consequently, if there are two tuples in R, such as t_1 = ('3-540-30153-4', 'Amazon', \$76.46) and t_1 = ('3-540-30153-4', 'Allbooksweb', \$126.99), $t_1 \succ t_2$ holds. Based on the database modeling approach, their proposals can seamless integrate with data formed in traditional database model, but hard to adapt to the preferences' change as context changes. In addition, the relational database-based models are often constrained in reasoning and inferring new knowledge on the user preferences, or rules definition. Particularly in mobile service setting, some researchers [HK04] proposed to store and manipulate user profiles in database. For each application, each user can configure their preferences called as *situation*. Each situation will be uniquely associated with the user-id. This approach takes into account the effect of context on user's preferences, i.e. each context property can be defined as an attribute in the *situation*. However, for each user, it is necessary to set up a new *situation* for each new application. Thus, it is still poor in dynamically employing user profiles for service personalization.

In the artificial intelligence community, user preferences are generally defined as preference relations using numeric utility functions and the results are ranked by the values of the utility function [AW00] [HP04]. Recommender systems and decision systems widely employ this quantitative modeling approach and benefit from computing the utility values, particularly with the multidimensional composition of preferences. More complicated utility functions can be elaborated by referring to users' behavior history and other users' collaborative choices. However, the utility functions are obviously sensitive to any change of the applications so that it makes it hard to transform this approach into a dynamic services setting.

Another potential proposal is logic-based modeling approach motivated by its strong expressiveness and reasoning/inference capability. It mainly embodies efforts from two different logics communities, i.e. Datalog/Prolog and Description Logic. As it is well-known, Datalog and Prolog support strong reasoning and data management functionalities, in particular they can easily combine with logical programming to make deduction and expressing rules [GJM01]. In [CCC$^+$04], Calì et al. proposed a DL-based framework to allow the profile description to be partially incomplete and addressed an algorithm to illustrate how to match the query considering the user profile. However, existing logic-based approaches, either Datalog-based or DL-based ones, fail to provide a sufficient solution to express the preferences in context-sensitive dynamic environments.

Doubtless, the description-logic based preferences can be easily transformed into ontology. In the CRUMPET project, the user preferences are described in an ontology. Each preference is defined as an attribute and closely associated with the properties of a given service. It shows the functionality of user preferences in matching certain services, e.g. tourism, but did not explain how the user preference ontology can be reused in new applications.

6.3 A Framework of User Profiles Interactions in LBS

Similar to the way that data profiles interact with core ontology, the user profiles need to communicate and further interact with the core ontology. At the syntactical level, the concepts and values, rules in user profiles need to be valid and consistent with the definition of LBS. At the semantic level, LBS will identify their semantics and apply the knowledge of LBS to set up the links between the context and services in core ontology and given user profiles. As stated before, issues concerning system architecture are not the foci of this dissertation. Therefore, we only show the part of the LBS infrastructure that conveys the communications between inter-related modular data in the core ontology (i.e. service, context, spatial, temporal, and user) and user profiles at the user side as follows:

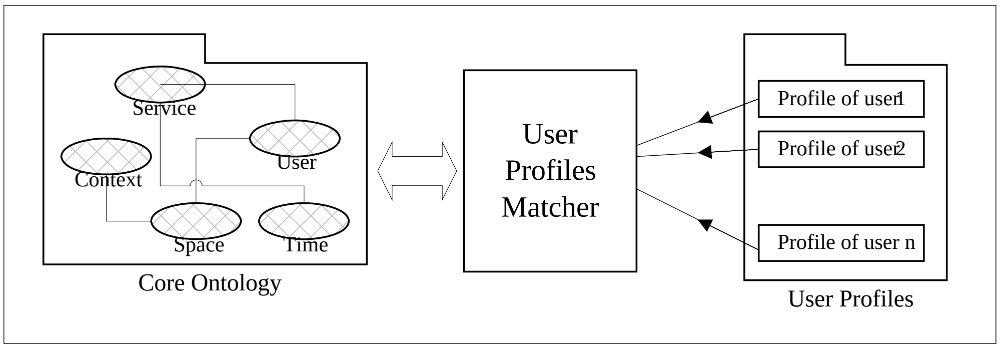


Figure 6.1: The Infrastructure of Communications Between User Profiles and LBS.

The left of the figure shows the different modules of the LBS core ontology and their associations. They provide the semantic support of concepts and values' data-types in user profiles. For instance, the user module may include frequent concepts in user profiles, such as age and profession and interest, and context module can include the information about local CollegeCalendar which is used in user profiles. The right of the figure displays a set of user profiles in LBS. We assume that each user has a single *complete profile*, which can be decomposed into several *contextual profiles*. As an intermediary between LBS and users, the *User Profiles Matcher* transforms user profiles into the knowledge understandable by the core ontology and then applies the knowledge in the core ontology to build up the links between the services and a given user. In particular, while a certain query is delivered to the LBS, it can assist the LBS to quickly identify what user profiles information is relevant to the given query and then look up what special user information is helpful to access to a certain service.

6.4 User-relevant Concepts in Core Ontology

In the core ontology, we have a set of built-in concepts to describe the characteristics of the LBS users. These concepts and their relationships form the user module. Each user class is characterized by a set of properties and constraints. Usually, users in one class share common characteristics or hold similar preferences on certain services. Each service class can be associated with one or multiple user class(es), which means services in class Service can suit for a user class in the *User Module*. User classification may become particularly useful when the user profile lacks specific preference information on a certain service. Preferences in this case can be taken from the specific user class the user belongs to. For instance, assume a user profile only contains his/her basic information, such as the *age* with the value "66-years-old", the *profession* with the value "retired", and the *health-status* with the value "common, with heart-sickness history". When the user is looking for a restaurant for dinner, LBS will identify the user's profile and generally categorize him/her into the *senior user* class in the user module. Using the association between *restaurants* and *senior users*, the LBS can potentially recommend the user some restaurants in a quiet environment, other than some brasseries or noisy music-bars. In this section, we explain how to define the user classes in the user module and then illustrate how to associate the user classes with the services in the core ontology.

In addition, some relations defined in chapter 3 can be properly applied to relations between user concepts. For instance, IsA ($\sqsubseteq$) can be used to express the relation between GraduateStudent and Student; disjoint can be used to relate Junior to Senior. Regarding the relations between inter-module concepts, e.g. the relation between user class and a service class, we will discuss them at the end of this chapter.

Each user class in core ontology is a subclass of class User, characterized by a set of properties and constraints, and defined in terms of axioms. The user classes are initially defined by the LBS experts, and are subject to change to adapt to the interactions between local services and users. It is worth noting that it is necessary to consider the local context when defining the user classes and their restrictions. For instance, in Switzerland, the term senior denotes people older than 65-years-old, but in France, the term senior denotes people older than 60-years-old. Let us assume that the user class Senior is a class in the user module, and it represents a group of people who are older than "65-years-old", and whose profession is "retired". In terms of axioms, the class "*Senior*" can be defined as follows:

$$\mathsf{Senior} \sqsubseteq \mathsf{User} \sqcap \exists \mathsf{Age}.{\geq_{65}} \sqcap \exists \mathsf{Profession}.\{\mathsf{Retired}\}.$$

The properties in the above axiom, such as *age* and *profession*, and similar static properties such as *gender* describe general information about the user. They are often regarded as a kind of factual data, and will not frequently change as time and space change. According to the profiles specified by a given user, each user can be associated with one or multiple user classes.

6.5 The User Profiles in Our Approach

The user profile is the key to customize services and to obtain tailored information. Two users asking the same request at the same location and at the same time should have different answers according to their profiles. This is the main functionality of user profiles in the majority of personalized information services. Furthermore, to adapt to the dynamic context, for the same user, depending on where, when, how, with whom, and why users are navigating in a physical space, his/her profiles and needs will vary (e.g. at home, at work). Therefore, profiles in LBS are characterized by their dynamicity and context-sensitiveness, contrasting with the fixed or application-coupled profiles mainly used in web services.

6.5.1 Contextual User Profile vs. Complete User Profile

In our work, each user has a *complete user profile*, which contains three aspects of the information, i.e. the user's characteristics, preferences and distastes, and user-defined rules. The first part stores the information that is inherently related to the user, therefore often static and context-insensitive, i.e. their values usually do not change as context changes, e.g. gender or nationality. The second part, preferences and distastes, clearly suggests what the user may like or dislike, and may include a ranking among the preferences and distastes. The last part allows the user to define specific rules for information delivery and selection and privacy protection, e.g. *Do not recommend any service provider which has a customer rating less than 90%.*

Several elements in the user's profile can be context-dependent, i.e. their specification and value may change from one context to another. The most elementary context-dependence is the one that makes the value of a property vary according to the value of some context item. For example, the value of ”my favorite sports” in the user profile may be different for the various seasons (spring, summer, autumn, winter). To use the favorite sport information, the LBS will need to know which is the current season. This knowledge comes from the context module.

A given context element may influence several choices in the user profile. The kind of activity pursued by the user is a typical example, as it may influence several specifications such as profession, interest domains, spoken languages, and relevant club memberships. Extracting sub-profiles from the complete profile according to these influential context elements generates a set of *contextual profiles*. Examples are illustrated in Figure 6.2, where the ”role” currently played by the user determines different contextual profiles, one for when the user in on a tourist trip, one for when the user is on a professional trip, and on for daily life use. The figure shows that, for example, the interest attribute is valued as (art, culture, hiking, cinema) in the tourist profile, while it is valued as (database, ontology, semantic web) in the professional profile. It also shows that different contextual profiles do not include the same attributes, e.g. the income attribute is included in the tourist profile but not in the professional profile. A contextual profile restricts the search for personalization to the elements it contains. For example, the professional profile in Figure 6.2 specifies that, should the user be in a professional context, the only elements that are to be considered for query personalization are user's profession, age, interests, languages and memberships. Our working hypothesis is that contextual profiles are defined by the user, both in terms of which ones to create and which content they should have. The definition of the required contextual profiles specifies for each one the predicate that the LBS has to evaluate to determine if the current context of a user query corresponds to a specific contextual profile. For instance, assume the user wants to define one contextual profile for when she is on vacation and one for when she is out for a meeting. If we further assume that users are characterized by a role $\mathsf{hasActivity}$ to an $\mathsf{Activity}$ class, the first context is selected if the role currently leads to the $\mathsf{Vacation}$ subclass of the $\mathsf{Activity}$ class, while the second context is selected if the role currently leads to the $\mathsf{Meeting}$ subclass of the $\mathsf{Activity}$ class:

$\mathsf{Context1} = \mathsf{hasActivity.Vacation},\ \ \mathsf{Context2} = \mathsf{hasActivity.Meeting}$

The predicate to select a contextual profile can be fully based on information that is specific to the user (i.e. the current user situation), but may also involve generic knowledge in the user module and knowledge in the other modules (context, space, time), in particular knowledge about the current context (as in the prior example on sports preferences). Links between user profiles/user module and context module may therefore be established by the LBS to enable contextualized personalization. Such links will be discussed in Section 6.6.2 later in this chapter.

Complete and contextual profiles may be organized into a variety of data structures. For instance, contextual profiles may be linked by subtype relationships, as in:

$\mathsf{hasActivity.EuropeanVacation} \sqsubseteq \mathsf{hasActivity.Vacation}$

$\mathsf{hasActivity.SummerVacation} \sqsubseteq \mathsf{hasActivity.Vacation}$

Which data organization to adopt for best management of user profiles is a relatively marginal issue, as long as the functionality to support contextualized personalization is available. User-defined rules are part of the specifications that may be contextualized. For instance, they may be associated with certain contexts, e.g. to reject any incoming message irrelevant to the activity at meeting, or to always ask for user's permission at online-shopping.

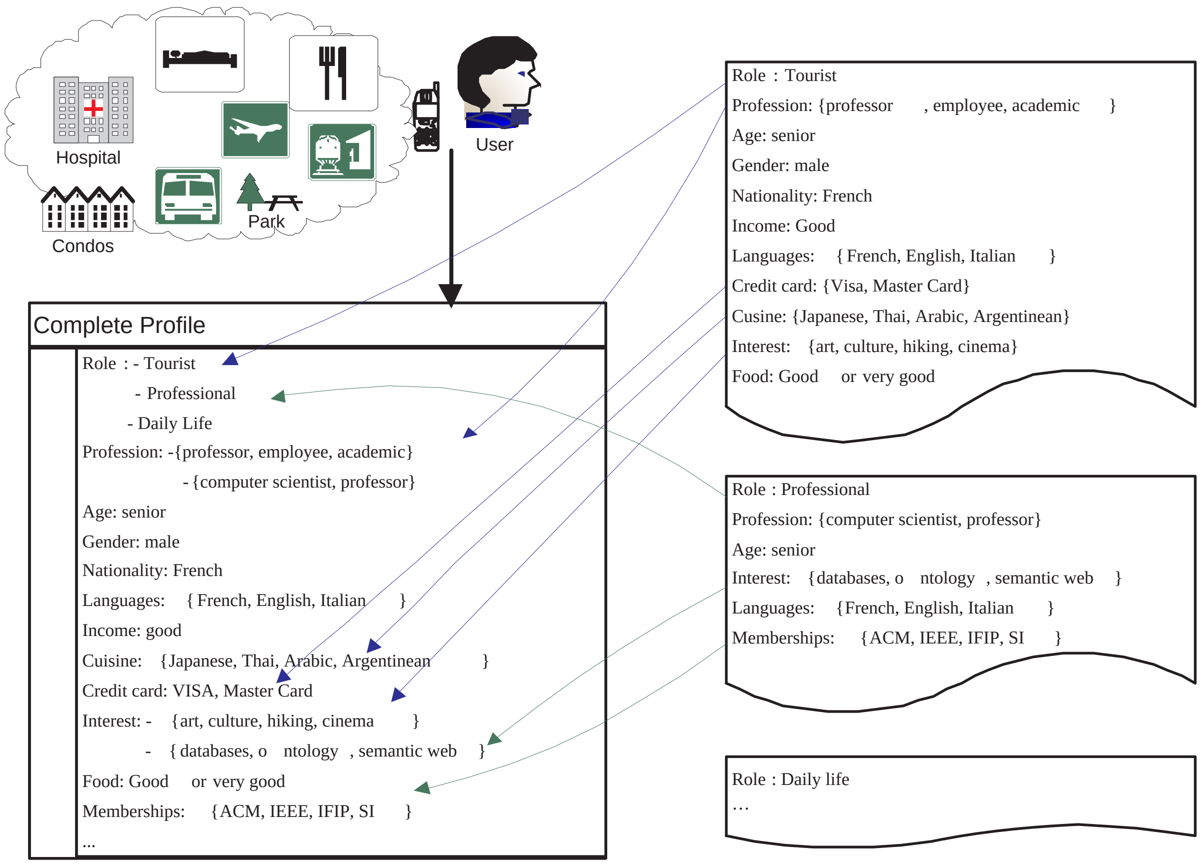

Hospital
Condos
Park
User
Complete Profile
Role : - Tourist
- Professional
- Daily Life
Profession: -{professor, employee, academic}
-{computer scientist, professor}
Age: senior
Gender: male
Nationality: French
Languages: {French, English, Italian}
Income: good
Cuisine: {Japanese, Thai, Arabic, Argentinean}
Credit card: VISA, Master Card
Interest: - {art, culture, hiking, cinema}
- {databases, ontology, semantic web}
Food: Good or very good
Memberships: {ACM, IEEE, IFIP, SI}
...
Role : Tourist
Profession: {professor, employee, academic}
Age: senior
Gender: male
Nationality: French
Income: Good
Languages: {French, English, Italian}
Credit card: {Visa, Master Card}
Cusine: {Japanese, Thai, Arabic, Argentinean}
Interest: {art, culture, hiking, cinema}
Food: Good or very good
Role : Professional
Profession: {computer scientist, professor}
Age: senior
Interest: {databases, ontology, semantic web}
Languages: {French, English, Italian}
Memberships: {ACM, IEEE, IFIP, SI}
Role : Daily life
…

Figure 6.2: An Example of user profiles for a given LBS user.

Though the contents of two contextual profiles for a given user may be different or overlap, each content always is a subset of the corresponding complete profile. Each complete user profile is identified by a unique user-id, and all its contextual profiles inherit this user-id. Once the user profile is created, this user-id is taken as the unique identification to store and manipulate the user's information (e.g. personal information, preferences and/or queries' log), as well to access certain services (in particular, some services may have certain constraints on the accessibility privilege to end-users).

The content of user profiles can be explicitly provided from user's input, implicitly learned from the user's behavior and information navigation history, or alternatively derived from the profiles of other users in the same interest group. The issues on acquisition of user profiles, in particular intelligent techniques for learning user preferences, are out of the scope of this dissertation. Instead, we concentrate on modeling and applying the user profiles in information delivery process of the LBS. We assume that initially the information in the complete profile is supplied by users. It can evolve as the user profiles are applied in LBS searching and/or shared with other users' profiles. As stated, the contextual profiles are also defined by LBS users, for instance, to give a *professional profile* and a *tourist profile*. Alternatively, they can be extracted from the complete profile of the same user by the LBS, by analyzing the properties in the user profiles and corresponding services, or referring to other users' profiles with the same contexts.

The complete user profile and the contextual user profiles may be formally defined as follows.

DEFINITION 6.1. Contextual User Profile. *A Contextual User Profile is a tuple cUP describing the user from the viewpoint of a certain context: cUP = (uID, $c_\heartsuit$, {(P, I)}, R), where uID is the unique identifier of the user, $c_\heartsuit$ is a context characterization, (P, I) is a set of pairs <property, instance> and R is a set of rules defined by the user.*

DEFINITION 6.2. Complete User Profile. *A Complete User Profile is a tuple UP = (uID, CUP) where uID is the unique identifier of the user, CUP = {cUP(uID, $c_{\heartsuit 1}$), ..., cUP(uID, $c_{\heartsuit i}$)}. For the given user uID, $\forall$ cUP(uID) $\in$ CUP.*

The definition shows that for each user the complete profile is actually the union of the contextual profiles. Each contextual profile is characterized by a defining predicate $c_{\heartsuit i}$, whose evaluation determines if the profile applies or not, and is composed of a number of properties with corresponding instances and a number of user-defined rules. Property sharing among the contextual profiles is via the name of the property.

In the previous chapter we have seen that a data profile is composed of a set of service profiles. Here we have seen that a complete user profile is composed of a set of contextual profiles. Despite this similarity, there are three evident differences between data profile and complete user profile, and between service profile and contextual user profile:

- A user profile describes an individual user; a data profile describes a data source and conveys the information concerning one or many independent services. Any data profile can be clearly divided into a set of independent service profiles; not all user profiles can be easily divided into disjoint contextual user profiles.

- Each service profile is explicitly defined within the data profile; a contextual user profile can be generated by the LBS learning from the user's query log and other users' profiles. Thus, the difference of information's origins may result in the uncertainty of the information, and some values in user profile are subject to change as the user has more interactions with LBS.

- The user profile allows the user to specify a set of rules, in order to personalize information delivery, to express privacy concern (e.g. if the service provider's rating is less than 90%, don't show them), or to represent more complex preferences/distastes (e.g. if no vegetarian restaurant is available, only recommend a Sushi Shop). Rules in service profiles are of a different nature. They come as hasCondition properties expressing the preconditions to access to the service.

- Both contextual user profiles and service profiles are context-sensitive. However, more often, the same type of services are sensitive to the same context, e.g. most rental prices for holiday apartments are similarly sensitive to the season; the users may have different concerns on same type of services (preference) in same context, e.g. in summer, user A prefers outdoor-sports but user B may prefer indoor-sports even if both A and B have interests in sports.

By comparing the data profiles and user profiles, we find out user profiles demand some new modeling constructs wrt the data profiles. Concretely, the new challenges concentrate on defining new types of properties, enhancing the dependencies between profile properties, and keeping the user-defined rules consistent with each other.

6.5.2 Classes

The user module in the core ontology gathers all classes related to the description and categorization of potential users, aiming at providing sufficient semantic support for generic personalization processing. For instance, the user module holds a seniorUser class defined as a restriction on the User class, the restriction consisting in the condition that only users whose age is greater or equal a given age (e.g. 65 years in Switzerland) are members of the subclass. Similarly, a Colleague class may be defined to express that a person X is a colleague or person Y iff X and Y work for the same employer. The user module can also be augmented with inference rules, such as

$(\text{SeniorUser} \sqsubseteq \text{User} \sqcap \exists\text{Age}.\geq_{65}, \text{User} \sqcap \exists\text{Age}.\geq_{65} \sqsubseteq \text{RetiredUser})$

$\Rightarrow \text{SeniorUser} \sqsubseteq \text{RetiredUser}$

While the user module deals with abstract users and user categories, user profiles are meant to provide concrete information (i.e. instances) about specific users. To be operational, information in the user profiles must somehow correspond (be mappable) to the classes in the user module (in the same way as information in the service profiles must be mappable to clases in the service module). For example, in the user module, the semantics of class Family can be well-defined, e.g. *a primary social group including parents and children.* However, in a concrete user profile, family may be enumerated as a set of people and their respective relationship with the user. This Family information in user profiles can be applied in identifying who has the rights to share and exchange information with this user. For instance, the user can specify "always expose my location to my family". Similarly, the user can define other classes if necessary, e.g. who are my colleagues, which companies (or service providers) are confidential enough to share the personal information etc.

$\mathsf{Family} = \{(\mathsf{Peter}, \mathsf{father}), (\mathsf{Mary}, \mathsf{mother}), (\mathsf{Shirley}, \mathsf{sister})\}$

$\mathsf{Colleague} = \{\mathsf{Fabio}, \mathsf{Nadine}, \mathsf{Lucy}\}$

$\mathsf{ConfidentialServiceProvider} = \{\mathsf{Amazon}, \mathsf{eBay}\}$

$\mathsf{User}(?x) \wedge \mathsf{ServiceProvider}(?y) \wedge \mathsf{Trust}(?x, ?y) \Rightarrow (?y = \mathsf{Amazon} \vee ?y = \mathsf{eBay})$

As we discussed, user profiles are often relevant to or dependent on certain contexts and/or services, e.g. to specify some context-dependent preferences on sports. Consequently, to better communicate with LBS, user profiles either are equipped with these concepts, or ensure that they exist in the core ontology or external ontologies. For instance, the preference/rule *"At rush-hour and when I am driving, inform me of the traffic ahead."* contain several context concepts, such as rush-hour, driving, and traffic. In this case, we assume the core ontology embodies the definitions of these context concepts and is able to obtain the relevant context data from certain context repository.

6.5.3 Properties

Properties provide crucial information about a user, her/his physical characteristics, e.g. age, profession, or health, and her/his preferences or distastes for specific things or services. Physical characteristics help the LBS identifying the basic features of a given user, leading to a possible classification into some user category, and then recommend some services by referring to other users who have similar physical characters and therefore are in the same category. For instance, any of a priori knowledge, similarity techniques and learning mechanisms may lead to associate the *Senior* class (category) to a preference for less intensive sports. Preferences and distastes help the LBS to personalize service matching according to the user's explicit specifications. Hence, we specify three categories of properties in user profiles: hasPreference, hasDistaste and hasCharacteristic respectively. They help defining other sub-properties as follows:

$\mathsf{hasPreference} \subseteq \neg\ \mathsf{hasCharacteristic}$

$\mathsf{hasPreference} \subseteq \neg\ \mathsf{hasDistaste}$

$\mathsf{hasCharacter} \subseteq \neg\ \mathsf{hasDistaste}$

$\mathsf{hasAge} \subseteq \mathsf{hasCharacteristic}$

$\mathsf{hasSportPref} \subseteq \mathsf{hasPreference}$

In Chapter 3 we have discussed some typical properties for the service module, i.e. *Composite property, Range Property, Dependent Property* and *Multi-resolution Property.* These types of properties can also be properly applied in representing some properties in user profiles. For instance, property *languageCapability* can be expressed as a composite property, i.e. composed of sub-properties *language* and *level.* The definition on *dependent properties* can be extended to define the context-sensitive user preferences. The multi-resolution property can be tailored to constrain different abstractions of information to be exposed to information sources with diverse confidentialities etc.

Considering the specific concerns from users' viewpoint, e.g. personalization and privacy control, we propose some new constructs in expressing these features. Hereinafter we define these constructs to support representation of various kinds of user properties , namely context-sensitive properties and privacy-sensitive properties. First, however, we add some semantics to multivaluation of properties.

6.5.3.1 Order-valued Properties

According to the classification in [PSZ06], a property can be either monovalued or multivalued. A monovalued property corresponds to a single value, e.g. age. In contrast, a multivalued property associates with a collection data, which can be a set, a bag or a list. For a property that holds more than one values it may be the case that these values are mutually equivalent with respect to any service that may call for the property. For instance, a user profile may have a CreditCard property whose value is Visa and MasterCard. It may be that for this user it is indifferent to choose either credit card. In this case, we call the CreditCard property is defined as a *mutual-value property*, which can be easily represented in OWL using the owl:unionOf axiom.

An alternative to mutual-value properties are *ordered-value properties.* These are used whenever there is a preference order among the values of the property, e.g. the user prefers value a to value b for the property. For instance, a user can express his/her preference on different cuisines, such as CuisinePref = {Chinese, Japanese, Thai}. In this example, "Chinese" is the first element in the set, meaning that Chinese cuisine is the favorite cuisine for this user; the Japanese cuisine is regarded as the second favorite, and so forth. Similarly, we can assume the same user profile has another property CuisineDistastes which corresponds to a list of values {canned-food, fast-food}. This means that the user has major distaste for canned-food, followed by a somehow lighter distaste for fast-foods. Consequently, both canned-food and fast-food will be excluded from the result of food service selection, but if this makes the selection empty a fast-food service may be returned. The definition of ordered-value property can be stated as follows:

DEFINITION 6.3. Ordered-value Property. *An Ordered-value Property is a property p(d, τ), with domain d and range the set of values $\tau = (t_1, t_2, \ldots, t_i)$ and i≥2, such that $t_1 \succ t_2 \succ \ldots \succ t_i$, where $\succ$ is a preference relation, and $t_1 \succ t_2$ means prefer t_1 to t_2.*

Example 6.1. The aforementioned user's preferred cuisine can be expressed as p(uID, τ):
CuisinePref $\subseteq$ p,
τ= (Chinese, Japanese, Thai),
Chinese $\succ$ Japanese $\succ$ Thai.

In this example, the domain d is the set of identifiers of a user profile, it suggests that the user with uID has such an ordered-value property. For instance, when the user is looking for a restaurant, it is useful to filter out restaurants the user does not like or has less interest in.

6.5.3.2 Context-sensitive Properties

In reality, it is common for users to change their preferences as the context changes for different reasons. Hence, if a property has different values in diverse contexts, we call it a *context-sensitive property.* For instance, one can like sushi but dislike to take it for breakfast. In contrast, if a property always has a unique value whatever the context is, we call it a context-insensitive property. To better express context-sensitive property, we give its definition as follows:

DEFINITION 6.4. Context-Sensitive Property. *A Context-Sensitive Property, denoted $p^{\prec c}$(d, τ), with domain d and range τ, is a property such as τ=(c, b), where c is a context component and b is a set of values for the property $p^{\prec c}$, and $c \rightarrow b$ holds (i.e. b depends on c).*

Example 6.2. The user's sports' preference changes as the context changes, e.g. the user likes mountain-sports and hiking in summer, but she likes ski and yoga in winter:

hasSports $\subseteq p^{\prec c}$,
hasSports(uID, ("summer", {hiking, mountain-sports}))
hasSports(uID, ("winter", {ski, yoga})).

The example shows that the property hasSports is a subtype of context-sensitive property and has two known context values, summer and winter. Since the two contexts belong to temporal contexts, the property hasSports can be regarded as a *time-sensitive* property. Similarly, should the context be in GeoContext, the property can be seen as a *space-sensitive* property. For instance, users having different credit cards associated to different account in different currencies may wish to define that different types of credit cards have to be used in different locations, e.g. use *American Express* in the USA, but use VISA in Europe.

In Definition 6.4, the context component in τ is not constrained to be a single context. Instead, it can be a combination of several contexts, which together determine the property value. A typical example is spatio-temporal dependency as discussed in chapter 4. It is similar to the multi-valued dependency in the database. Hence, in this case, the functional dependency can be written as $\forall p^{\prec c}(a_1, ((c_1, \ldots, c_j), b_1)), p^{\prec c}(a_1, (c_1, \ldots, c_j, b_2)) \Rightarrow b_1{=}b_2$, *we say* $(C_1, \ldots, C_j) \Rightarrow b$. Let us see an example where multiple types of contexts together determine the property Interest.

Example 6.3. The user's sports' preference changes as multiple contexts change, e.g. the user prefers hiking or outdoor-swimming in summer with family, but chooses more risky sports like rock climbing or wind surfing in summer vacation with friends:

hasSports $\subseteq p^{\prec c}$,
hasSports(uID, ((”summer vacation”,”with family”), {hiking, outdoor-swimming}))
hasSports(uID, ((”summer vacation”,”with friends”), {rock climbing, wind surfing})).

Example 6.4. A temporal context sensitiveness: The user's interests in professional work and vacation are different:

hasInterest $\subseteq p^{\prec c}$,
hasInterest(uID, (”profession”, {database, ontology})),
hasInterest(uID, (”vacation”, {art, museum, cuisine})).

For instance, if the user registers to a new book reminder service, the professional interests can direct the LBS to choose database/ontology relevant books for the user. In contrast, when the user is on vacation, the nearby art museum or exhibition will be chosen as most attractive for this user.

6.5.3.3 Privacy-sensitive Properties

Privacy is a crucial concern in most e-services. In social life, people incline to protect their personal information, e.g. the birthday, the family information and other contact, from disclosure to unknown people. With the growing popularity of e-services, more and more users are customized with some online-payment services. Consequently, privacy protection and trust-related issues have become the main challenge. How to effectively protect and manage privacy and personal data is a common focus for most service providers and users, and has to be taken into account in LBS. In the LBS setting, we assume the user stores information in the mobile device and uses it to communicate with the LBS. LBS need to provide effective means to manage and protect the user's privacy. The user shall be given the ability to configure what information can be shared with which service-providers, and in what context. For instance, the users may intend to share her personal data only with certain types of services she is strongly interested in. We define the property isPublicTo to suggest to whom the user is willing to expose her profile data.

DEFINITION 6.5. Privacy-sensitive Property. *A Privacy-sensitive Property is a property $p_!(I, C^U)$, in the user profile, with domain* individual *and range a class C^U, such that I is a property value in the user profile, and C^U refers to information receiver such that $C^U \sqsubseteq C^{user} \sqcup C^{service-provider}$.*

The definition allows information receivers to be either service providers or other users. The latter assumes that some mechanism for transferring data from one user to another is supported by the LBS. This could be for instance a personal archiving space as usually provided by communication services on the web (e.g. Google, MSN, Yahoo). We do not elaborate any further on this mechanism, which is not essential to LBS.

Example 6.5. The user can configure the privacy protection using privacy-sensitive properties. For instance, the user can specify that credit card information can only be exposed to Amazon and PayPal service providers.

Credit Card $\equiv$ {Visa, Master}, Trusted- Provider $\equiv$ {Amazon, Paypal},
{eBay} $\in \neg$Trusted-Provider,
$p_!$(VISA,Trusted-Provider) $\Rightarrow$ $p_!$(VISA, Amazon), **NOT** $p_!$(VISA, eBay).

When a property value is specified as privacy-sensitive, the LBS knows the user only wants to share the information with the specified users/providers. Hence the LBS needs to request user's permission when some new/unknown users or providers somehow request access to this information. Privacy-sensitiveness can also be associated to a user profile as a whole. In this case, all information inside the profile will only be available to the specified users/services. Conversely, it can be tailored to finer levels of detail: For instance, the birthdate property can be completely disclosed to friends and family, but to other users only disclosing the year of birthdate is allowed. Notice that privacy protection is under full control by the users, there is no initiative to be taken by the LBS in this domain.

6.5.4 Dependencies and Rules

In chapter 3, we have defined the generic dependencies, which can be appropriately applied in any module, as well as in user profiles. For instance, the property nationality and languageSpoken are linked by a dependency if it is the case that users are assumed to be fluent in (one of) the official language(s) of the country she is a citizen of. If the two properties are explicitly specified in the user profile, the dependency over their values needs to be satisfied.

As also discussed in chapter 3, a specific kind of dependency is derived attributes. As an example in the user profile, the age property depends on user's birthdate and the current date. It is a kind of static temporal dependency: Every year the user's age will increase regularly by "one". In this case, the property is dependent on the time-stamp of the input.

Rules in user profiles are categorized into two types: user-defined rules and derived rules. User defined-rules only apply to the user who defines them. They can be changed anytime by the user. Derived rules can either be defined a priori by LBS designers (e.g. the rule inferring spoken languages from nationality) and apply to multiple user categories and even all users, or be derived by LBS on the basis of the observation and analysis of a single user's history of queries and behaviors and then applied to the user or to her user categories. For instance, in the field of web information retrieval, many efforts have been devoted to implicit user profiling by analyzing user's bookmarks, web-pages navigation, and references to collaborative peers. Through analyzing the user's queries, inference of user preferences can be tailored to certain service types in certain locations, times and contexts.

6.6 Connections and Mappings between Modules

In chapter 3, we have proposed to structure the LBS's knowledge base in terms of modular ontologies. Rather than maintaining a single ontology, the modular ontologies strategy enables LBS data management to benefit in the following aspects: to ease the maintenance and reasoning of the evolving ontologies, to enable ontology reuse and sharing, to guarantee consistent semantics between modules thanks to the modular ontology languages. Up to now we have described each module independently, and how they can be created and maintained. However, making intelligent use of the modules to support query personalization and contextualization calls for carefully established interconnections between the modules. This section discusses such interconnections.

In literature, ontology languages for module interconnection can be categorized into two main types: Distributed Description Logics (DDL) and ε-Connections approaches. Both approaches aim at linking existing ontologies seen as modules of a larger modular ontology that is available to users. The two approaches differ in the semantics and the purpose of the links. The DDL framework enables to couple multiple ontologies using a set of *bridge rules*. A bridge rule is an axiom involving elements from two ontologies. The axiom has one of the following formats: $C_i \sqsubseteq C_j$, $C_i \sqsupseteq C_j$, and $a_i \rightarrow a_j$, where C_i and C_j are classes respectively from *ontology i* and *ontology j*, a_i and a_j are two individuals from *ontology i* and *ontology j* separately. Bridge rules are intended as a mechanism to import data from one module into another module. This approach seems interesting for the semantic web paradigm, since it allows to develop ontologies in an autonomous yet interoperable fashion, in particular through reuse of existing ontological elements. Although C-OWL has been criticized [GPS06] for lack of sufficient reasoning support, e.g. the undecidability issue for certain subsumption relations, its reasoning capabilities have recently been extended so that Tbox reasoning is fully supported. Rather than reuse through import, the ε-Connections approach aims at connecting existing distributed ontologies that represent different yet complementary domains. In the ε-Connections approach the concepts from different modules are expected to show little or no overlapping. Instead, they can be correlated so that knowledge in one module can be enhanced with related knowledge in another module.

In our work, the core ontology is modularized into three modules, i.e. *Context Module, Service Module* and *User Module*. The three modules describe different aspects of the information necessary in LBS. They can autonomously evolve, but need to hold consistent semantics so as to be inter-operated and reasoned by the LBS at the logical/semantic level rather than via the syntactic modularity provided by the construct *owl:imports* in OWL. The concepts in the different modules of our core ontology are connected by inter-module roles, which we call *linkTo property* as they are similar to the link property of the ε-Connections approach. For instance, a linkTo property may connect some *SkiResort* service to the *Summer* element in the context module to state that this service is available in summer. While using an interconnection approach close to ε-Connections, we do not want to tie this to excluding concept overlapping between the modules. We believe it is preferable, for greater flexibility in the design of the modules, to allow that the various modules share some common concepts, i.e. concepts having same definitions and relationships. For instance, the service module may include a subclass of *exhibition* services for exhibitions on '*Science and Nature*'; in parallel, the user module may include a subclass of users interested in '*Science and Nature*'. In this case 'Science and Nature' is the common (i.e. overlapping) individual between the two modules. Slightly changing the example, we can illustrate broader commonality between the modules: For instance, in the Service module, a **hasTopic** role links the class **Exhibition** to the class **Topic**, and in the User module, a role **hasInterest** links the User class to the same **Topic** class.

Besides the three modules of the core ontology, the service profiles and user profiles can be regarded as external ontologies. The service profiles matcher and the user profiles matcher guarantee the semantic consistency of the profiles with the concepts and roles defined in the core ontology modules, yet these external ontologies hold their own individuals. The Figure 6.3 visualizes the overlapping of the core ontology modules and their relationships with external ontologies.

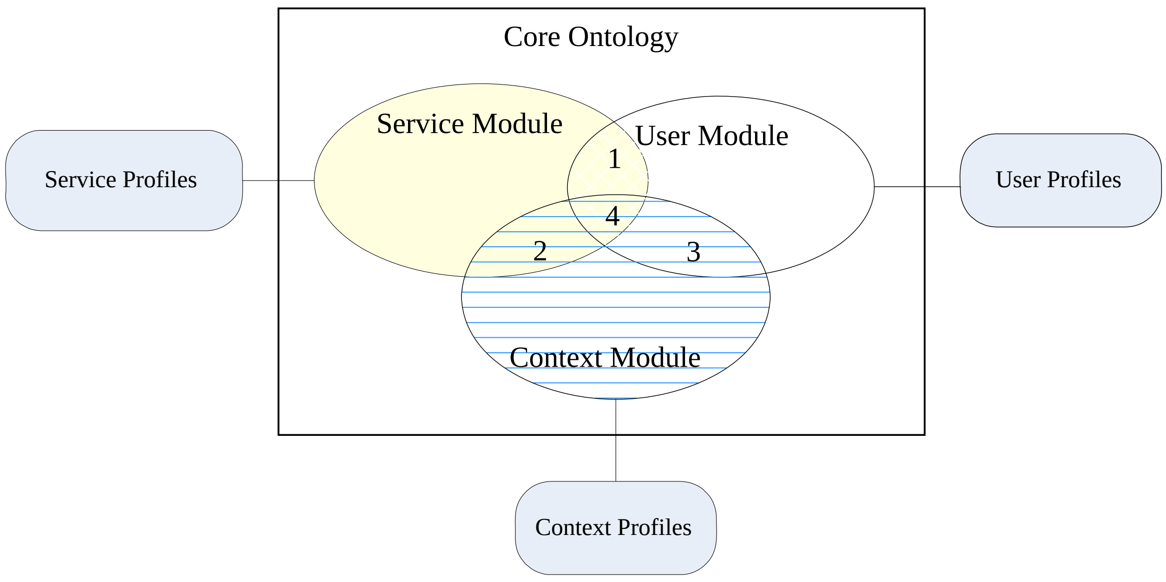


Figure 6.3: The Modularization of Concepts and Relations in LBS.

The above figure shows the disjoint and overlapping regions among the diverse modules of LBS. Region 1 represents the elements common to the service module and the user module, e.g. the concept **Topic** and the individual 'Science and Nature' in the example. The concepts SkiResort and Season are disjoint, respectively residing in *Service Module* and *Context Module*. Regions 2 and 3 separately stand for the common elements between service module and context module, and between user module and context module. Region 4 represents common elements of the three modules. For instance, the concept Interface may be common to the three modules. A *User* may use the concept to specify the preferred format of query results in terms of "text" or "image". The context class *Device* can be characterized by a property *hasInterface* representing the device's display capability such as *Text* or *Image* in black-white/color. The *Restaurant* service may be able to show its information in terms of text descriptions with/without images.

In the following we first introduce the *link properties* and ε*-Connections Modularity Ontology* according to [GPS06]. Next, we present the C-OWL approach to represent subsumption between concepts in different modular ontologies. Then we introduce the *linkTo property* that we use within our LBS to connect concepts in the diverse modules of the core ontology. In addition, we present how to define the relevancy between properties of different modules, in order to achieve the interrelation and orchestration of the three modules. Finally, we discuss how to define and discover the relevancy between classes/properties of different modules.

6.6.1 ε-Connection and C-OWL

ε-Connection and Link Properties. An ε*-Connection* is a kind of distributed ontology that is composed of a set of ontologies, where the component ontologies describe disjoint parts of the real world, and each one is an OWL-DL ontology extended with link properties that link the ontology to some other component ontologies. The component ontologies are called ε*-Connected ontologies.* "An ε-Connected ontology K contains a sequence of annotations, axioms and facts. Annotations, as in OWL, can be used to record authorship and other information associated with the ontology, including imports" [GPS06]. Besides the information about the same kind of constructs as in an OWL ontology (i.e. classes, object properties, axioms, etc.), an ε-Connected ontology contains the information about link properties which allow crossing between different ontologies. These link properties are roles that link one class of an ε-Connected ontology to a class of another ε-Connected ontology. Using the syntax and definition in [GPS06], link properties are defined by the following axiom:

axiom ::= 'Link(' **linkID**['Deprecated'] { **annotation** }
  'foreignOntology(' **OntologyID** ')'
  {'super('linkID')'}
  {'domain('**description**')'}
  {'range('**foreignDescription**')'}
  ['inverseOf('**linkID**')']
  ['Functional' | 'InverseFunctional']
axiom ::= 'EquivalentProperties('linkID linkID { linkID } ')'
  | 'SubPropertyOf(' linkID linkID ')'

Different from the object properties in OWL ontology, in an ε-Connection, all *link properties* must be explicitly declared. Each link property is declared in the ε-Connected ontology that contains its domain (i.e. its source ontology). Its target ontology (called the foreign ontology in the axiom above) must be specified. A link property cannot be transitive or symmetric, but it can be equivalent to or sub-property of another link property. Maximum cardinality constraints (equal to one) may be defined like for an OWL object property. As with OWL object properties, link properties can be used in axioms that define restriction classes. In an ε-Connected ontology O, axioms may use all the link properties whose source ontology is O. Restrictions may use the universal (allValuesFrom), existential (someValuesFrom), value (hasValue), and cardinality constructs. The definitions of restrictions are given as follows:

restriction ::= '**Restriction**(' **linkID linkRestrictionComponent** { **linkRestrictionComponent** } ')'

linkRestrictionComponent ::= 'allValuesFrom(' foreignDescription ')'
| 'someValuesFrom(' foreignDescription ')'
| 'value(**ForeignIndividual**(' individualID '))'
| cardinality

An ε-Connection is composed of a set of ε-Connected ontologies. Thus, an ε-Connection can be regarded as the union of distributed and linked ontologies, each of which may describe a single domain. Notice that the *link property* itself, like ordinary ontology roles, does not convey any semantics other then linking elements from two ontologies.

C-OWL and Bridge Rules. A C-OWL ontology is another kind of distributed ontology. It is also composed of a set of OWL-DL ontologies, but the component ontologies describe possibly overlapping parts of the real world. The component ontologies, called local ontologies, are linked together by inter-ontology is-a links, called *bridge rules.* A bridge rule asserts that a class (or role or individual) of a local ontology describes the same set (or a sub-set) of real world phenomena as a class (or role or individual) of another local ontology. The interpretation domains of the local ontologies are related together, by a set of binary relations, one binary relation per couple of local ontologies. These binary relations link together the individuals from the various ontologies that describe the same real world entity. A bridge rule from a class (or role) of a local ontology O1 to a class (or role) of a local ontology O2, is an is-a link modulo the binary relation that links the interpretation domain of O1 to the one of O2. Referring to [BGvH+03], bridge rules and mappings between ontologies in C-OWL are defined as follows:

> "A *bridge rule* from i to j is a statement of one of the four following forms,
> $i{:}x \xrightarrow{\sqsubseteq} j{:}y,\ i{:}x \xrightarrow{\sqsupseteq} j{:}y,\ i{:}x \longrightarrow \perp j{:}y,\ i{:}x \longrightarrow * j{:}y,$
> where x and y are either concepts or individuals, or roles of the languages L_i and L_j respectively."
>
> "*Furthermore, given a OWL-space* $< i, O_i >_{i \in I}$*, a* Mapping M_{ij} *from* O_i *to* O_j *is a set of bridge rules from* O_i *to* O_j*, for some i,* $j \in I$"*.*

As local ontologies describe overlapping parts of the real world, they may be conflicting, but also knowledge may be propagated from one local ontology to another one. Depending on how two local ontologies are related by bridge rules, the subsumption (is-a links) may be inferred from one ontology to the other one.

Conclusion: These two proposals, E-Connections and C-OWL, present two basic ways for linking distributed ontologies: 1) by is-a links between classes describing (at least partly) the same real world phenomena, 2) by inter-ontology roles linking classes that describe different real world phenomena.

In our work, the ontology modularization strategy is not merely motivated by easing the ontology maintenance and sharing. More importantly, it helps to discover and specify the relevancy of concepts (and properties) between various modules. In the sequel, as a query is delivered to the LBS, the LBS modular ontologies can understand how to employ the knowledge in the User and Context modules to answer the query, so as to make the LBS personalized and context-aware.

6.6.2 LinkTo Relationships in LBS Modular Core Ontology

Building on the definition of link property of ε-Connections, we refine it to suit our LBS settings. Recall the definition in chapter 3, we define that a *modular class* is a tuple *C = (c, module, axiom)*, holding the class name, the module name, and its defining axiom. For the discussions in chapter 3, it was not necessary to specify if the relationships it introduced were inter-module or intra-module relationships. In this thesis, if we do not explicitly define a relationship as an inter-module relationship, it is an intra-module relationship, i.e. the two classes involved in the relationship belong to the same module. Given a link property, its classes must belong to different modules, hence it is an inter-module relationship. The Condition property in Chapter 4 is an example of inter-module relationship, as it links service data to user data to describe which category of users may access a service. For instance, a student movie service can be only available to students, i.e. the user should hold a valid student card. To distinguish link property in ε-Connections from our refined definition, we call the latter a *LinkTo Relationship*, described as follows:

DEFINITION 6.6. LinkTo Relationship. *A LinkTo relationshipis a DL-role* $\overrightarrow{l}(C_a, C_b)$*, where* $\overrightarrow{l}$ *is the name of the role,* C_a *and* C_b *are two module classes,* $C_a = (c_a, module_a, axiom_a)$ *and* $C_b = (c_b, module_b, axiom_b)$*, that respectively are the domain and range of the role, and the following holds:*

- $module_a, module_b \in \{Module_{Service}, Module_{Context}, Module_{User}\}$ *and* $module_a \neq module_b$*;*
- $C_a \sqsubseteq \neg C_b$*,* $C_a, C_b \in CO$ *(Core Ontology);*
- $\overrightarrow{l}(C_b, C_a) \notin \overrightarrow{L}$*, where* $\overrightarrow{L}$ *denotes the set of LinkTo relationships.*

In this definition, we explicitly constrain the modules of the classes within *Service, Context, User* modules. The LinkTo relationship can not be transitive or symmetric (i.e. if $\overrightarrow{l}(C_a, C_b) \in \overrightarrow{L}$, $\overrightarrow{l}(C_b, C_a) \notin \overrightarrow{L}$). In particular, the two classes involved in LinkTo relationship are disjoint and both are included in the core ontology, which ensures the decidability and exactness of reasoning. In addition, to differentiate intra-module relationship and inter-module relationship, classes C_a and C_b must belong to different modules.

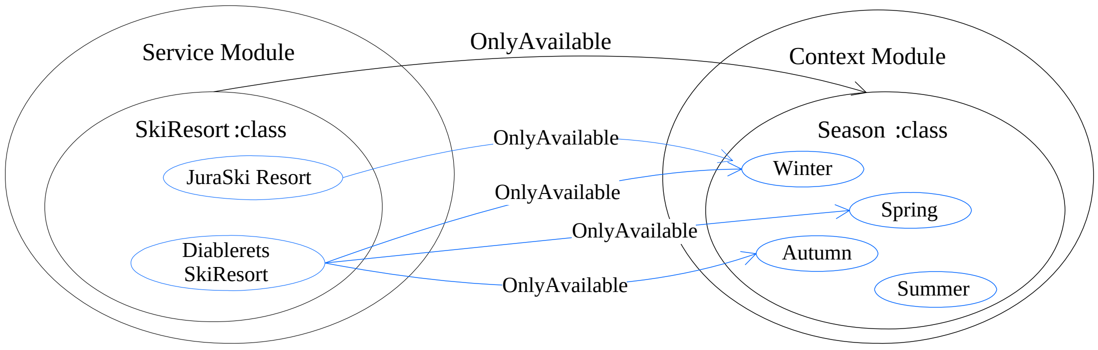
6.6.3 Determine Relationships

In LBS, it is common that a module class C_a is determined by another module class C_b, rather than the simple semantics "Link C_a To C_b". Determining links are particularly useful in filtering out unavailable services. For instance, in Switzerland, only few ski resorts are open in summer. When a user asks for a ski service, the context Season is deemed as the determinant factor. To enforce the *determinant* semantics, we introduce the *Determine* relationship as follows:

DEFINITION 6.7. Determine Relation. *A Determine relationship is a DL-role $\overrightarrow{d}(C_a, C_b)$, where $\overrightarrow{d}$ is the name of the role, C_a and C_b are two module classes, C_a = (c_a, $module_a$, $axiom_a$) and C_b = (c_b, $module_b$, $axiom_b$), that respectively are the domain and range of the role, and the following holds:*

- *$module_a, module_b \in \{Module_{Service}, Module_{Context}, Module_{User}\}$ and $module_a \neq module_b$;*
- *$C_a \sqsubseteq \neg C_b$, C_a, $C_b \in CO$ (Core Ontology);*
- *$\overrightarrow{d}(C_b, C_a) \notin \overrightarrow{D}$, where $\overrightarrow{D}$ denotes the set of Determine relationships, and $\overrightarrow{D} \sqsubseteq \overrightarrow{L}$;*
- *For each individual $b_i \in C_b$, there must exist an (or a set of) individual(s) $\{a_1, \ldots, a_i\} \sqsubseteq C_a$. iff $\exists x$, $x \in \{a_1, \ldots, a_i\}$, b_i becomes valid.*

Example 6.6. A *Determine Relationship* between Ski Service and Season Context.
SkiResort $\in C^{Service}$, Season $\in C^{Context}$, $\overrightarrow{d}$(Season, SkiResort):



We have $\overrightarrow{d}$("Winter", "JuraSkiResort"), $\overrightarrow{d}$({"Spring", "Autumn", "Winter"}, "DiableretsSkiResort").

For individuals in Service Module Class SkiResort, "JuraSkiResort" and "DiableretsSkiResort", and individuals in Context Module Class Season= {Spring, Summer, Autumn, Winter}, the *determine* relationship $\overrightarrow{d}$ expresses the context *Season* is one of the *necessary conditions* to access the service *SkiResort*. More precisely, *JuraSkiResort* service individual is only available if current Season is *winter*. The service individual DiableretsSkiResort is available in all three seasons, Spring, Autumn, and Winter.

Similarly, the *determine* relationship can link classes respectively from service module and user module.

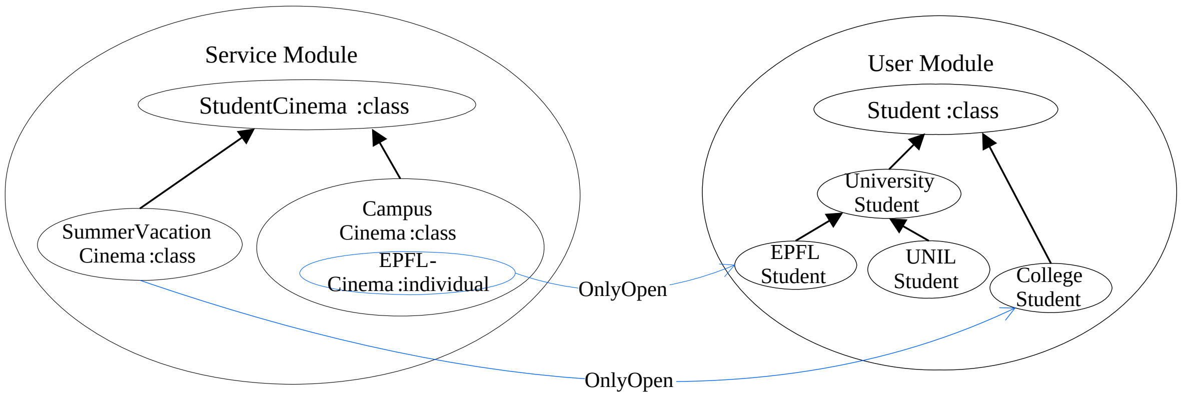


Figure 6.4: An Example of Determine relationship between service module and user module.

$\mathsf{Student} \sqsubseteq \mathsf{User} \sqcap \exists \mathsf{RegisteredAt.}(\mathsf{College} \sqcup \mathsf{University})$,

$\mathsf{CollegeStudent} \sqsubseteq \mathsf{Student}$, $\mathsf{EPFL\text{-}Student} \sqsubseteq \mathsf{Student}$, $\mathsf{UNIL\text{-}Student} \sqsubseteq \mathsf{Student}$.

$\mathsf{StudentCinema} \sqsubseteq \mathsf{Cinema} \sqsubseteq \mathsf{Service}$,

$\mathsf{CampusCinema} \sqsubseteq \mathsf{StudentCinema}$, $\{\text{EPFL-Cinema}\} \in \mathsf{CampusCinema}$.

$\overrightarrow{d}(\mathsf{Student},\ \mathsf{StudentCinema})$,

$\overrightarrow{d}(\mathsf{EPFL\text{-}Student} \sqcup \mathsf{UNIL\text{-}Student},\ \text{"EPFL-Cinema"})$;

$\overrightarrow{d}(\mathsf{CollegeStudent},\ \mathsf{SummerVacationCinema})$.

The above example states that the class EPFL-Cinema in the service module is linked by a determine relationship instance to the class EPFL-Student in the user module. It means the user must be a student registered in EPFL to access the EPFL-Cinema service.

6.6.4 Influence Relationships

In the two previous examples, the target class defines a strong pre-condition to accessing the services. If the pre-condition is not satisfied, the service will not respond to service requesters. In reality, there also exist some less strong dependency relations between classes from different modules. This weak dependency means the source can not determine the availability or exactness of the target, but its value can have an effect on the quality or metric of the target. For instance, the network connection usually has an influence on download services. This type of influence represents a dependency between *download service* and *network context*, but it is not a determinant factor as described in the determine relations. To represent weak dependencies we propose an Influence relationship as follows:

DEFINITION 6.8. Influence Relationship. *An Influence Relationship is a tuple involving two classes C_a and C_b, denoted as $\overrightarrow{i}(C_a, C_b, axiom)$, where C_a and C_b are two module classes, $C_a = (c_a, module_a, axiom_a)$ and $C_b = (c_b, module_b, axiom_b)$, and* axiom *stands for the influence/effect of C_a on C_b, such that the following holds:*

- *$module_a, module_b \in \{Module_{Service}, Module_{Context}, Module_{User}\}$ and $module_a \neq module_b$;*
- *$C_a \sqsubseteq \neg C_b$, C_a, $C_b \in CO$ (Core Ontology);*
- *$\overrightarrow{i}(C_b, C_a)$ is a role in CO, $\overrightarrow{i}(C_b, C_a) \in \overrightarrow{I}$, where $\overrightarrow{I}$ is the set of Influence roles,*
- *$\overrightarrow{i}(C_b, C_a) \notin \overrightarrow{I}$.*

Example 6.7. An Influence relationship between context NetworkConnection and service OnlineFilm:

$\overrightarrow{i}(\mathsf{NetworkConnection},\ \mathsf{OnlineFilm},\ \mathit{axiom})$

$\mathit{axiom}{::}\ \mathsf{OnlineFilm} \sqcap \exists \mathsf{hasQuality.}_{=}\{\text{Good}\}$

$\Rightarrow (\mathsf{NetworkConnection} \sqcap \exists \mathsf{hasSpeed.}_{\geq}\{\text{100Kbps}\})$

The example above states an Influence relationship between the *online film* service and the *network connection* context. The axiom suggests that the service can provide *good* quality when the connection speed is faster than 100Kbps. When the connection speed is below this value, the quality of the online film service may become unsatisfactory. To explicit the influence relation, more information can be added to the axiom in the Influence relation, e.g. what connection speed is required to provide same good quality online film service with high resolution of the video. In reality, the axioms may be more complex, and can be defined and improved by some experts, but these axioms are not the main focus of our work.

A given service may be influenced by a set of contextual concepts, e.g. an outdoor sport, such as bicycling, may be influenced by the weather and the traffic. In such a circumstance, the LBS will look for all concepts associated with this service by *Determine* and *Influence* relationships. Then the LBS will match them against the relevant information either in the context module or in the user profiles. We further discuss this as part of the query matching and processing issues in chapter 8.

Similarly, *Influence* relationships can be employed to link the service module and the user module, e.g. a given service may *attract* a set of groups of users. For instance, a science exhibition can be recommended to students and to families with children. This type of recommendations are based on the experiences or observations of some domain experts. For instance, some recommendations on local cultural events can be defined by some local tourist professionals. The application of the *Influence* relationships in the LBS is motivated by transferring the context/user knowledge to the service processing, so as to make LBS more knowledgeable.

6.6.5 Property Relevance Associations in LBS Modular Ontologies

The *Determine* and *Influence* relationships denote a strong or weaker dependency between two module classes. Consider, for instance, the impact of network throughput on download operations. This can be described as a dependency relationship from the NetworkConnectionclass the Download service class, and its subclasses such as MovieDownload and MusicDownload. However, the dependency can be more precisely described referring to the properties of the involved classes, namely the properties that rule the dependency. In the example, the property hasSpeed of the context class NetworkConnection is the one that influences the values of property hasQuality of all Download service classes. We therefore introduce the concept of a dependency between properties from different modules. Such a dependency is illustrated in Figure 6.5:

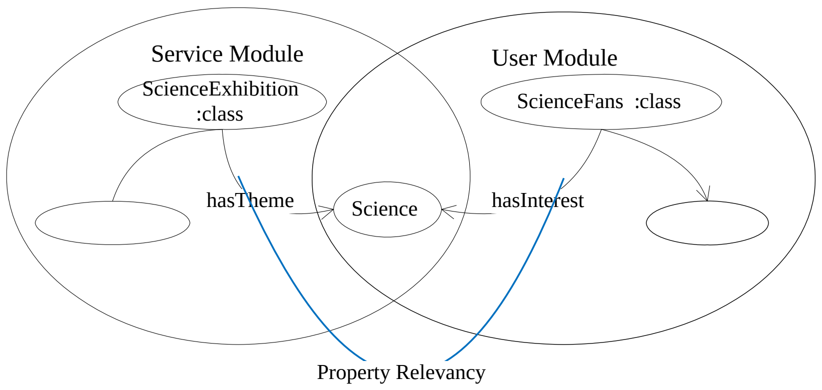


Figure 6.5: An illustration of properties relevancy between different modules in LBS.

In this example, we see that both the user class ScienceFans and the service class ScienceExhibition are associated with the concept Science through respectively the hasTheme and the hasInterest properties. We can infer that the ScienceExhibition can be interesting to ScienceFans. However, without an explicit relationship, it is not evident to discover the interaction between the two classes and between the two properties hasTheme and hasInterest. We propose the following construct to specify the interactions between properties that in different modules correspond to the same or overlapping concept or data-value.

DEFINITION 6.9. Property Relevance Association. *A Property Relevance Association is a tuple $\widehat{r}(p_a, p_b, axiom_r)$, where p_a and p_b are two properties, $p_a = (C_{a1},\ CVD_{a2})$ and $p_b = (C_{b1},\ CVD_{b2})$, C_{a1} and C_{b1} are classes, CVD_{a2} and CVD_{b2} are either classes or value domains $axiom_r$ defines the dependency between p_a on p_b, and the following holds:*

- $C_{a1},\ CVD_{a2} \in module_a,\ C_{b1},\ CVD_{b2} \in module_b,$
- $module_a,\ module_b \in \{Module_{Service}, Module_{Context}, Module_{User}\}$ *and* $module_a \neq module_b;$
- $p_a,\ p_b \in CO$ *(Core Ontology);*
- $CVD_{a2} \sqcap CVD_{b2} \neq \perp.$

The semantics of the association is that whenever the intersection between CVD_{a2} and CVD_{b2} is non empty for a specific individual $\alpha_1 \in C_{a1}$ and a specific individual $\beta_1 \in C_{b1}$, the two individual are considered as a matching pair in the query processing strategy.

Example 6.8. A Relevance Association between hasTheme and hasInterest properties:
p_a:= hasTheme, C_{a1}:= Exhibition, CVD_{a2}:= Theme,
$individual_a$:= SwissCulture,
p_b:= hasInterest, C_{b1}:= User, CVD_{b2}:= Interest,
$individual_b$:= EuropeanCulture,
AND $axiom_r$: $CVD_{a2} \sqsubseteq CVD_{b2}$
$\Rightarrow$ $\widehat{r}(p_a,\ p_b,\ axiom_r)$, $\widehat{r} \sqsubseteq \widehat{R}$, where $\widehat{R}$ is the set of Property Relevance Associations.

In this example, the class C_a ('SwissCulture' exhibition) belongs to *Service* module; the user class C_b defines a group of users who have interest in *EuropeanCulture*. In the core ontology, both *SwissCulture* and *EuropeanCulture* are subclasses of *Culture*, and there exists SwissCulture $\sqsubseteq$ EuropeanCulture. According to our Definition 6.9, we can conclude there is a relevance association between the two properties p_a and p_b. This construct is specially aimed at discovering the intrinsic relevancy of information in diverse modules for query processing and personalization. In other words, given a user's profile(s) and a set of services, it facilitates LBS task to filter out the services irrelevant to a given user and respond to the request in a personalized fashion. When the value of property p_b or p_b changes, the relevancy can not hold. For instance, if the exhibition is still concerning *SwissCulture*, but the user is interested in *AsianCulture*, the relevancy between two individuals (i.e. 'SwissCulture Exhibition' and 'AsianCulture Fans') does not hold since SwissCulture is disjoint with AsianCulture. In general, the relevancy construct can be illustrated as follows:

Discussion: In this section, we have given our definitions on inter-module connections. Both the Determine relationships and the Influence relationships are specified at the meta-level by the LBS and its experts. For

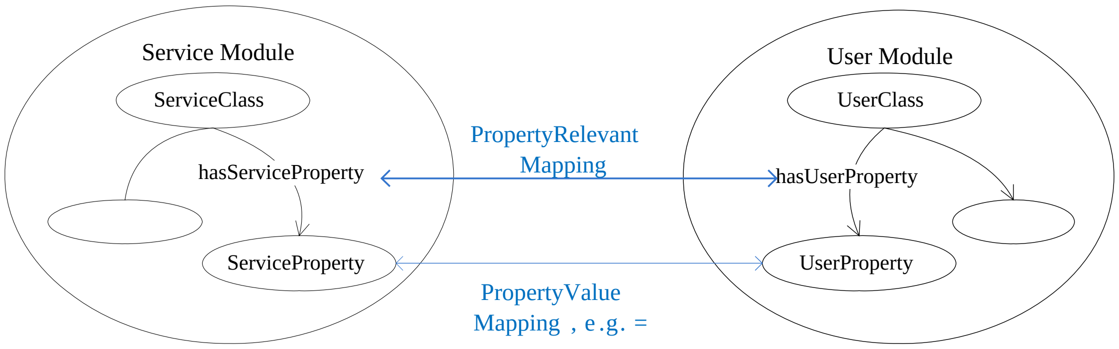


Figure 6.6: An illustration of properties relevancy between different modules in LBS.

instance, from the service to context, they can provide information if a service is restricted or favored by one context or a set of contexts. From the service to the user, they suggest if a service is only accessible to users with a specific profile, alternatively, if a service is potentially favored by a group of users. Definitely, the determine and influence relationships can be instantiated. For instance, a service profile is determined/influenced by the constraint in a given user profile (e.g. Shirley does not like all action movies). In contrast, the user may specify specific interest/dislikes for a given category of services, e.g. Shirley likes Kentucky but not MacDonald in the fast-food service category.

In contrast, the *Property Relevance Association* is defined in a bottom-up manner. In other words, the LBS will look at the similarity/relevancy between target concepts (or target data values) of two properties, as shown in Example 6.8. If two properties are associated with equivalent/non-disjoint classes (or data values), the LBS will create the association between the two corresponding properties. Notice that in our definition we specify the source class of the property, instead of a global property. This suggests that a property of a given class C_a is associated to another property of a specific class C_b. It assists the LBS in discovering the potential relevancy between property instances separately in two modules.

6.7 Chapter Summary

This chapter is twofold. The first part analyzes user profiling. We first discussed some related work from an over-abundant literature that is mainly characterized by solutions tailored to the needs of some specific application. We then presented our LBS framework in which user profiling materializes on the one hand at the data level, as a set of specific user profiles, and on the other hand at the meta-level, as an abstract and generic description of user categories. The latter forms the user module in our LBS modular ontology. The main contribution in this respect has been to make very explicit that for each user a number of different profiles may hold, each one with a specific content and globally characterized by a qualifying predicate that allows selecting for each query the user profile that is suitable for the query. We called these the contextual user profiles, as their selection stems from considering the current context, a combination of user-related context and generic context. We also introduced some further characterization of properties that are particularly useful in defining user profiles.

The second part of the chapter capitalizes on the fact that description of the Service, Context, and User modules has been achieved, to discuss inter-connections between the modules of the modular ontology. These interconnections are essential to LBS. They allow the LBS to perform contextualized personalization. This is supported by the inter-module relationships between services and context elements that are relevant to them, between users and context elements that determine their context-sensitive properties, and between services and users to support user preferences as well as to express access limitations to services dedicated to specific user categories. These inter-module connections transform a set of dedicated ontological modules into an integrated and consistent knowledge infrastructure leading to the intelligent LBS that we aim to characterize.

Chapter 7

Query Reformulation

7.1 Introduction

The curiosity for and pursuit of information and knowledge is the driving force that makes humans more and more capable and intelligent to make use of surrounding resources and solve complex problems. In addition, to express their needs is an innate capability of humans. For instance, children can intuitively ask questions and know the world from their parents. At school, students can learn and accumulate their knowledge by communicating and discussing with teachers and classmates. In a library, users can inquire the librarian or check the index to find a book. By asking questions and exchanging information, humans can gradually learn new things, and obtain the specific information they need.

Whatever means one may rely on, queries need to be precisely expressed in order to produce the right answer. Although in human-to-human conversation acquiring information by incremental interactions can be very successful, when we attempt to convert this idea to the communication between the computer and humans, it can become unexpectedly frustrating. One of the important factors is whether the computer-human interface is practical and efficient enough to help users to express the right query (i.e. query formulation). Another important factor is that the computer is unable to fully understand and interpret the query expressed in natural language, and it can hardly grasp the implicit context of the user's query, thus being unable to properly react to the requests (i.e. to perform query rewriting and answering in a contextualized and personalized way).

7.1.1 Query Formulation Techniques - An Overview

In a primer book 'Designing The User Interface' [SP05], authors identified five primary styles of effective human-computer interaction: direct manipulation, form filling, menu selection, command language, and natural language:

- In *direct manipulation* systems, the designers must fully capture all the tasks and user's actions at the system design phase and employ pointing devices to direct users to carry out tasks rapidly and observe the results immediately. This paradigm provides high subjective satisfaction for novice and intermittent users due to its easy learning, easy retention, and error prevention, but may pose great challenges for programming, in particular for complex tasks. For instance, data entry interfaces in many banking ATMs are based on direct-manipulation, providing users with a convenient way to quickly master the interface and smoothly finish a certain number of predefined tasks.

- In *menu selection* systems, the user is always provided one or more functional list(s). Having acquired some familiarity with the meaning of the terms in the lists, the user can have a complete and clear overview on all functions and then choose the appropriate item. Advanced features, i.e. allow users to choose and edit frequently/rarely used functions, can be offered to increase the freedom of the control on the interface; they are especially favored by frequent users.

- In *form filling* systems, the user is supposed to fill in the blanks of a form in order to get answers. This requires the user to understand what each blank means and how to fill it correctly. Form filling is suitable for both unsophisticated and frequent users. Although it may increase the possibility of errors occurrence and needs a tentative training, it is efficient and widely used for querying structured data, e.g. library information systems.

- In *command language* systems, the user must be familiar with the commands, becoming an expert user of the interface. Such expert users overwhelmingly prefer command interaction over other styles of interactions, mainly because it allows very fast specifications (i.e. typing a command code is faster than scrolling through menus) and support user's initiatives (e.g. creating ad-hoc macro-commands) and flexibility (no need for a taxonomy of commands). Conversely, because it usually takes long time to learn and master the commands, its high complexity undoubtedly discourages the novice users and non-professional users. The operations in UNIX systems are a typical example of a command language system and perfectly illustrate the split between expert and non-expert users.

- *Natural language* systems set minimal knowledge requirements on users, since the interactions are mainly based on arbitrary natural language sentences or phrases. Through dialogue and incremental interactions, the user's task may be identified and responded. However, the lack of context information easily leads to too many interactions, which may largely frustrate the user and produce rather unpredictable results .

This classification offers valuable principles to the system designers for choosing the right styles of interaction between the system and the users. However, it is a generic classification for whatever systems or domains. For a specific domain or task, for instance *information seeking and representation*, there exist more sophisticated alternatives. In [Hea99], Hearst investigated the features and effects of different query formulation techniques in each style. Besides the styles discussed in [SP05], Hearst explored other types of techniques used in query formulation interfaces, e.g. Boolean queries, faceted queries, and graphical interfaces. A *Boolean query* is specified by Boolean connectors and descriptive metadata. Although it is the current mainstream way in modern information access systems, it is exposed to the confusions of the Boolean operators for users, results' low-relevancy and poor-ranking capability. A *faceted query* is built up with a set of categories, each of which contains a different facet (topic or feature type). For instance, to look for a cook book, its faceted query may be the conjunctives of (vegetable OR sea-food), (main OR salad OR soup), (Chinese-cuisine OR Thai-cuisine), which implies the user is looking for a 'cook book' meeting all three facets. Faceted querying is gradually getting more interest given the following advantages: only most relevant facets for users are introduced in the query specification; it can improve the results' relevancy due to the usage of semantic disjunctives terms in the same facet; and its categorization is favored rather than that in clustering-based search, since the categories and concepts for faceted query are based on the good domain knowledge, instead of the similarity measurement of terms (words and phrases) in documents.

Regarding the styles of query formulation interfaces, we can observe two parallel taxonomies, i.e. form-based and purpose-based. The former taxonomy is characteristics of user inputting or choosing certain information pre-defined by the system, which can be regarded as data-oriented, e.g. query interface for databases. The latter taxonomy is largely motivated by the user herself/himself, which can be regarded as purpose-oriented popular in next-generation web search engines.

Actually, the form-based taxonomy has been widely discussed in the literature. It can be traced back to the earlier natural language (NL) interface initiated by the HCI community. In NL dialogue systems [JWS81], the natural language interface was employed in query-answering by either modelling human's conversation or using the human discourse as the system's backend. Consider, for instance, a question such as 'who is the original inventor of the printing press?'. According to its syntax, the question can be successfully parsed and formulated into: who + is + the original inventor of the printing press. But such a knowledge-base is often limited by its capability to answer complex questions beyond its predefined discourse.

Although NL query interfaces, be they textual or audio ones, strongly appealed to researchers and practitioners and have been widely covered in literature, whether it can eventually be widely and regularly used in real systems is still an open question. To lower the complexity of NL dialogue processing, restricted forms of NL were developed. They defined their own language syntax and semantics, and users must utilize these predefined terms to make querying, e.g. the assertional database language SQL (e.g. Select $\cdots$ From $\cdots$ Where $\cdots$) and the navigational one CODASYL (e.g. Find, Find Next, etc.).

The restricted format of NL interfaces offers much convenience and functional flexibility for frequent users, with its well-defined syntax. In addition, the query processing turns to be much more efficient than for queries in classical NL interface. However, it also brings heavy workload to users, as we have mentioned in the discussion of command language interfaces [SP05]. To explore the benefit from database languages and pursue the interface's universality and simplicity, other plausible query interfaces attracted more attention, i.e. from the keywords-based and concepts-based to menu-based styles.

The keywords-based style becomes more and more influential with the increasing popularity of Internet, and it has been widely used in Web search engines. Instead of sentences as in NL interfaces, users just input few keywords which are immediately matched against existing web documents retrieved by the web engine. The concepts-based style is more intelligent than the keywords-based one, because it applies concept-based matching instead of merely keyword matching techniques. The idea is to replace the keywords in a query with the concepts they actually represent in user's mind. Concept replacement may depend on the relevant knowledge repository, e.g. the supporting ontology. Distinguished from the query specification in database, in both above means, without the professional querying skills or schema information, the user can find the relevant information by incremental query refinement and navigation. The keywords-based styles have become full-fledged in the commercial market, such as Google, and Yahoo, while concepts-based style is still at the explorative phase. Aiming at completely overcoming the low-recall and poor-precision of answers in keyword-based search, the concept-based approach holds the potential to become the next-generation web search strategy.

Slightly different from both approaches above, the menu-based (list-based) means provide more complete structures and functionalities as supported by the system, i.e. each item in the menu corresponds to a possible query pattern, which are pre-established and organized in hierarchies. This style can largely release the user from lengthy input and possible errors, so that it is especially applicable for a simple/modest task or a limited domain.

All above styles of query specification unavoidably resort to single textual input. However, we cannot neglect the fact that there also exist other query specification styles based on visual means, for instance, the form-filling and direct manipulation styles. The form-filling style has been widely used in database systems, since its interface can be easily understood and mastered with shallow query-relevant domain-knowledge. In library and digital disciplines, to look for a book, the user just needs to fill out one or several properties about the book through the query interface, such as 'name', 'ISDN', 'author(s)', 'year of publication', or formulate more advanced queries with Boolean operators such as AND, OR, >, <, etc. Then, the user's query can be translated and formulated into a database query language (e.g. SQL, OQL, Datalog etc.) that could be fully understood and manipulated by the database system. However, more complex queries cannot easily be expressed by common end-users with the form-filling style, so that they need to be directly written in certain database definition languages. In addition, the majority of web services interface still choose the form-filling interface for users to specify their queries. To further simplify the query formulation and user intervention, the visual means can be directly employed, e.g. metaphor-based approach. The more advanced graphical means can be more semantic-powerful, such as degree-based and fisheye-based, which can limit the search space so as to improve efficiency of the query specification process.

Let us now investigate the second taxonomy, on purpose-based query specification. As studied by many researchers, general web search techniques often encounter the low-recall and low relevance problems. One of main reasons is that the goal of the user is not precisely or completely expressed in keyword-based web search so that the system cannot understand the need behind user's query, i.e. why the user makes the search [RL04]. For instance, assume the user's intentional goal is 'where can I find a Swiss chalet?'. If the input to the search is just 'Swiss chalet', the result can contain links on restaurants and hotels entitled with 'Swiss Chalet', even a huge amount of chalet history information. Some researchers proposed to improve the relevancy by rewriting the query with additional context information [Law00]. For instance, the user can first choose a category and then input the keywords so that the search space is limited within a specific category, e.g. Yahoo or CYC. In addition, users can directly ask a domain-specific website for special information needs, e.g. Amazon.com for a book, Monster.com for a job. To some extent, this can help to disambiguate the syntax of input terms within a certain domain.

Research reported in [Bro02] builds on analyzing the query log and surveying users to classify web search behavior into three categories: navigational, informational and transactional. More recently, an interesting finding from [RL04] shows that the majority of informational search attempt to locate a merchandise or a service rather than to learn about it. In particular, the mobile interface design must take careful attention on the importance of instant access to service information and on providing an easily operable process that considers all '7C' factors: context, content, community, customization, communication, connection and commerce. [LB03].

In LBS, query formulation is a prominent factor for user satisfaction in getting on-the-spot information needs fulfilled, since the mobile users may easily get frustrated if confronted with uneasy interactions with the system just for eliciting their queries. Unfortunately, the study on query formulation for mobile services, especially for LBS, has been largely ignored and substantially lagged behind the needs. In the following part, we investigate if existing styles satisfy the requirements in mobile services, as well the existing design made by previous LBS.

7.1.2 Query languages

From the beginning of relational databases and object-oriented databases in 90's, to deductive databases, database systems have provided well-developed tools to manage data. Each type of database must be built upon certain mathematical and logical theory and have an appropriate query language. For instance, SQL ('Structured Query Language') is deemed as the most popular and widely used data-manipulation and data-definition language for relational databases. OQL is designed to operate data in object-oriented databases, while the integration of relational databases and logic programming specially results in deductive database systems [Liu99]. As web sources become more and more available, people are getting used to seeking information from Internet and search engines. However, the metadata of web-pages and websites are still too limited to answer complex queries, and its essentially text-frequency based matching strategy is far away from the query processing capabilities in database. Thus, the W3C Consortium proposed RDF/XML as new standardization goals.

It is significant to choose a suitable query language to fully support query answering and certain reasoning capabilities in LBS. A viable query language for LBS should meet the following desiderata:

- It is generally acknowledged that it is a cost-expensive and time-consuming task to integrate and maintain all data instances in a single repository. In addition, the characteristics of data sources in LBS, i.e. heterogeneity, dynamicity and authentication protection make this task impossible. Hence, as the data core of the LBS, a LBS ontology possesses the knowledge about the basic organization and functionalities of all local data services, in terms of a set of data profiles, but the service instantiations in these data profiles are still autonomously stored and maintained at data services' sides. Hence, the query language should be flexible but intelligent to query this LBS ontology and find which data services can meet the current service request.

Table 7.1: A comparison of different query formulation strategies.

Styles	Advantages	Disadvantages	Feasibility and Adaptivity	Existing Practices
1. Textual NL-interface based (sentence-based)	Simple and universal for different-levels of users.	Hard to implement and maintain, error-prone, hard to formulate right query for end-users.	Almost impossible to widely apply in services with different domains.	no
2. Audio NL-interface based (voice-based)	a most native and favorite way of query expression for humans, especially favored more than the input on small keyboard.	1) the low-precision of recognizing in a mobile and possibly noisy environment, 2) the lack of a robust language model enabling quick recognition with a huge vocabulary [FM02], 3) the needs of special attention on location-specific speech recognition.	It is just at a tentative step rather than the commercial market. Its technical challenges (e.g. voice recognition in dynamic environments) constrain its popularity.	Google Lab has already explored to build up a prototype voice search system by creating and analyzing different language models from a huge set of trained queries.
3. Restricted (or Reformatted) languages based	well-defined language syntax provides the efficiency for query processing and the possibility to make complex queries	hard to learn, it becomes impossible without the schema-like information about the services, and the long statements are unsuitable for limited handheld display.	it fits for well-organized service structures, but users must be very familiar with the language syntax, the service structure and vocabulary. So it can not be universal or practical for different levels of LBS users.	no
4. Form-filling based	easy to learn, very specific for each type of service.	hard to implement the service-dependent forms, and have specific requirements on mobile device (e.g. running java or JSP etc.).	It is applicable for a certain domain, rather than the multiple-domains LBS.	it is popular in web services research community, but in real market it is only applicable in a unique domain, e.g. news subscription service on mobile device.
5. Keyword-based	simple and universal for different-levels of users, 2-3 terms input are acceptable for handheld device.	error-prone, possibly low-relevance, poor filtering and ranking with huge amount of answers.	It has been widely used in web search engines. But the low-relevance is a critical challenge for the limited mobile screen and capability. And it needs to be improved to represent and retrieve the spatio-temporal information from web-pages.	most search engines.
6. Concept-based	more intelligent keyword-based approach, high-relevance, help to understand the purpose of the query.	the maintenance of concept knowledge base, e.g. ontology repository.	It is a possible tacit in next-generation web search, as well, it can be applied in LBS if they can provide sufficient spatio-temporal and contextual concept support.	The project SEWASIE applied an ontology-enabled interface to query the data from a specific perception in the ontology repository. In particular, the project CRUMPET simply presented an tentative ontology-based approach in LBS setting. How to design the ontology-based querying interface on small mobile devices is still an open question.
7. Menu-based (List-based)	simple and universal for different-levels of users, less user's input and intervention, error-prevention.	hierarchy-structure limits the amount of terms in menu, and repetitive navigations between hierarchies may frustrate users.	It is specialized for certain services and interactions, rather than a large spectrum of services. Its ease to learn can appeal to the novice and modest users, for the experts it seems too rigid to quickly locate or to tailor for their frequent use.	i-area service at DoCoMo in Japan has been able to supply the information of nearby facilities to mobile users, which are organized in a multi-layered menu format. And the requested service is represented with functions and location.
8. Direct manipulation (Metaphor-based)	simple for different-levels of users, error prevention, and visual intuitive particularly for handheld devices.	sometimes, what information will be represented after the actions and navigations is vague for end-users. It is also hard to navigate between different categories. And it can require pointing devices to control the manipulation.	This approach is also very common in tour-guide or route navigation services. Users can benefit from its visual intuitiveness and ease to follow.	In GUIDE system, mobile visitors can navigate with a map in Lancaster City, and ask the attraction information by touching the info button when approaching to an object. The requests are presented as thumbnail type pictures with textual descriptions [CDM+00]. Similarly, TIP project employed the map- and text- based metaphor to guide and recommend the users with interesting sights [HV03].
9. Map-and-Hyperlink based	to combine the GIS support with the hyperlink of the service gives users more intuitive and comprehensive spatial information.	specific means is needed to identify the association between the location in GIS and spatial information on web-pages. And the query specification in spatial dimension is still limited.	It is practical to widely use but can have new challenges on spatial web-pages processing.	Yahoo! Local allows users to search a service name with free text, locating 'in' a place in terms of 'address, city or zip code ', and results are represented with a map and a list of relevant web-pages. The Froogle advanced search enables customers to specify the name, price, system-defined category and vicinity of the target product using free text, and answers with product's picture, price, ratings and hyperlinks.

- Built-in reasoning capabilities are desirable in the LBS query language. It is because data profiles in LBS are generally organized in hierarchies, but relational database query languages are poor to efficiently deal with hierarchical data, object-oriented languages can not handle the reasoning as relational query language, and logic query languages must be extended to provide sufficient support to complex data-types and property-types.

- The query manipulation on context-sensitive data is important in LBS. Besides the most typical context data, i.e. spatial and temporal constraints, other types of context-dependent knowledge can be potentially useful. For instance, 'near' in Beijing may mean 'within 5km', but in Lausanne it may mean 'within 1km'. In this example, the city scale turns out to be a piece of contextual knowledge useful for semantic query evaluation. In addition, we need to carefully investigate the generic and specific needs of query specification in the context dimensions. Rather than the terms/relationships in conceptual modelling, in LBS, the human-familiar ones (e.g. near, in the south of, expensive, calm) are preferred than some theoretical denotations.

- The accessibility of data sources to certain users can be partially or fully restricted due to privacy issues or commercial reasons, so that the data accessibility management on diverse data sources is needed, i.e. within the same category of services, identify which ones are accessible and to what level of detail or which part of services. In return, the user can configure which service can access to her/his profile information or even decide what information (e.g. in profile or search history) can be obtained and utilized by the LBS or data services etc. Hence, the query language should be aware of the accessibility constraints from both users and services sides.

7.1.3 Chapter Outline

The reminder of this chapter is structured as follows. Section 2 discusses the main challenges in query formulation for Location-based services, and then points out our contribution. Section 3 details our query formulation approach, in terms of <what, where, when, what-else>, that aims to provide a context-aware and concise query interface tailored for LBS requirements. Section 4 describes the formalization of translating the query from natural language format to RDF/OWL-based language that can be manipulated by the LBS, and also addresses the relevant constraint issues in query formulation. Section 5 concludes and points to the future work.

7.2 The Challenges and Our Contribution

The success and prevalence of search engines, such as Google and Yahoo, indicate that users have already adapted and learned to formulate their information needs in a free-text way and finally find out the information they ask for. But, the search fashion in conventional web can not be easily transferred to the handheld web, where human's reading means and the presentation of the content have to be largely changed to comply with limited display-capability [JMMN+99]. For instance, in the conventional web the user can easily browse a long web page by scrolling up/down, or open several web pages in parallel. But in handheld web, it is not practical to do the aforementioned operations. Instead, the original content needs to be concisely presented, possibly divided into consecutive pages and browsed by navigations. Thus, although handheld devices do provide unprecedented convenience for mobile users to access information anytime and anywhere, they pose new challenges in many technical fields, such as limited computation capability, wireless networking connection, privacy management, interface design, query specification and user's interaction mode with mobile devices etc.

Our work focuses on the query formulation and query processing in LBS. As discussed in the previous section, existing paradigms are either too simple or too rigid (e.g. list-based or metaphor-based), or unintuitive to provide little hints in understanding why redundant and overabundant answers return and how to refine the query (e.g. conventional web search), or posing too high requirements onto mobile device capability (e.g. to display the ontology concepts and relations in ontology-based style). We will address four of the major difficulties as our foci, in interface and query formulation issues:

- **User interface design: universal, concise and effective (FORM).**
  In the HCI community, to pursue a design-for-all fashion, a unified interface must satisfy two requirements: user-awareness and usage-context awareness [SS04a]. The user-awareness means interactions are tailored for each given user according to his/her abilities, preferences and requirements etc. The usage-context awareness refers to the fact that the interface can change to adapt to changes in the context. The adaptation can cover from the channel shift and modal change to sources selection, etc. However, while the unified interface design comes to the realization phase, it must overcome two technical obstacles, the diversity of the device capabilities and the efficiency of interface adaptation. Hence, is a free-text based interface a solution to fully satisfy the needs in LBS in terms of unification and conciseness and effectiveness? From the analysis of web search behaviors, it was observed that web users have customized and learned how to formulate the query in certain terms to avoid result sets with too many or too few results. This fact implies that users are customized to utilizing free-text based query formulation to locate and sort services and products. Starting from this experience, our design integrates another significant part in LBS, i.e. context, as the complement to the core of service in the query.

- **The contextual expressions (CONTENT).**
  In most approaches of query representation in service search, little attention was focused on the expression on contextual dimensions, e.g. location and time. For instance, the user cannot clearly express their goal of service search in a specific geographical range and within a certain time. In conventional web search, with input 'hotel near railway-station', most results present web-pages including keywords 'hotel' + 'near' + 'railway-station', departing from the semantic meaning of the original query. The situation occurs because in current search engines, on one hand, a unified textbox-like interface can not clearly disambiguate 'near railway station' as a spatial condition, on the other hand, current web information retrieval techniques are not efficient enough to do the location-based service search. To the

best of our knowledge, the only efforts on spatial conditioning of query expression are made in Yahoo! Local and Froogle advanced search at Google. But they just provide relatively simple spatial operands, such as 'in' a city, address etc. In addition, the query expression in the temporal dimension was never addressed.

- **Multi-level of Constraint controls - Hard, Soft or No constraints (WHERE).**
  When users seek a service or a product, they often hold some conditions or preferences on certain properties. The difference between the conditions and preferences is the constraint level, i.e. hard condition (the target service or product must satisfy the condition) or soft condition (the target service or product should preferably satisfy the condition). Although the modeling issues on multi-level of constraints have been intensively discussed in fuzzy and preference modeling in database e.g. [Kie02], how to represent such constraints in query expressions in LBS is still an open question. Our work is motivated by enabling users to simply expressing the constraints on different dimensions, in order to avoid no answers or redundant answers. The dimensions are referred as subject, spatial, temporal, and other features. When the user has no constraint on certain dimension, the constraint level drops down to zero. Accordingly, the hard constraint is scored as 1, and soft one is scored between 0 and 1.

- **Different abstraction levels of semantic on each dimension (SELECT).**
  In mobile search, content presentation of results is mostly predefined by the systems, they often ignore the fact that the user can only be concerned with certain properties of the service rather than the expectation from the system. For instance, in a mobile search for a hotel, the result is often associated with a link, a map and a price, but the user can have more concerns on the facilities and ratings from others. Hence, to allow users to specify what they want to see in the result can be an advantage for different needs of mobile users.

7.3 Query Formulation in our LBS

7.3.1 Our approach

In this section, we present a *tuple-like query* formulation approach to help users to express their queries in our LBS. It involves four dimensions, and it is simply formulated as a tuple as:

<what, where, when, what-else>

It can be understood in such a way as 'look for service of type *what*, that are temporally available in the *when* timeframe, that are geographically available in *where*, and mandatorily or preferably meet the conditions in *what-else*'.

In this tuple, the term what is used to specify the subject of the query, which could be a facility, a service, a product, or a general term. It can be expressed with a simple word or a small set of terms, e.g. 'tourist office' (a facility), 'car rental' (a service), 'mp3 player' (a product), or 'top 3 attractions' (a term). During query processing, what is eventually translated as Q_{what} which can be identified and well understood by the LBS as a Service, a Product or just a Term.

The term where explicitly delimits the spatial location or area of the requested service. If where is not specified, the default spatial specification is (near, current_location). Where can be further described by the sub-formula (spatial-relationship, spatial-argument(s)). Spatial relationships can be the traditional binary topological relationships, such as within(distance-argument)of(reference-argument), the sub-formula being written as (spatial-relationship, argument1, argument2) and so on. As discussed in chapter 2, since the manipulation and computation on geographical data will be mainly supported by an embedded or external GIS assistant, its real computation capability on spatial data and relationships largely depends on the GIS used by the LBS. We just focus on three types of spatial relationships frequently used in spatial query formulation, i.e. metric, topological, directional (See Table 3.2).

The term when emphasizes the user's temporal conditions on availability of the requested service. The default specification for when is (after, current_time). Similarly to where, when can be described by the sub-formula (temporal-relationship, time_parameter(s)). Binary temporal relationships, e.g. between, can be denoted as (temporal-relationship, time1, time2). We assume the LBS has a pre-defined ontology able to identify the basic temporal terms and operators, such as before, after, between etc. which have been elaborated in Table 3.1.

The term what-else explicitly specifies what other conditions the service must (may) satisfy for user's current request. The conditions can be simple predicates on properties of the targeted service, such as *price* $<$ *400 CHF* for services selling digital cameras, or predicates on the existence of some specific qualifications of interest to the current request, e.g. services 'offering a student-discount or membership discount'. What-else is an optional component of the query, composed of zero, one or more predicates. The predicates can be unary ones, such as haveDiscountFor(student), and binary predicates such as operator(attribute, value), e.g. <(price, 400 CHF). The basic operators include $<$, $>$, $=$, $\leq$, $\geq$, $\neq$.

Naturally, we assume the user input for what-else is consistent at both the syntactic and semantic level. For instance, assuming Q_{what} is 'hotel', examples of incorrect predicate specifications in $Q_{what-else}$ include the following: 1) the predicate (price, =, 'blue') is syntactically incorrect as the value domain for 'price' is either numeric or a qualitative value in a predefined list such as *"high", "moderate", "cheap"*; 2) the combination of predicates (price, <, 50CHF) AND (price, >, 60CHF) is obviously incorrect as the two predicates are contradictory; and 3) the combination of predicates (price, =, 'cheap') AND (category, >, 'four star') is syntactically correct but semantically incorrect (incompatible) as the LBS knows that five star hotels cannot be cheap. In such cases, the LBS will prompt the user to correct the erroneous specification; in return, if the user fails to correct the contradictory parts, the LBS will just ignore the contradictory predicates in further query formulation and processing steps.

7.3.2 Subject dimension - what

The subject dimension is the core of the query, directing the LBS to look for such a service 'what'. Recall query formulation in relational database, where queries are expressed as SQL statements, i.e. in the format

Select ... *From* ... *Where* ... , before the query is parsed by the query compiler and optimized. Similarly, regardless of conditions in other dimensions, the subject dimension must be firstly parsed and looked up with the LBS ontology. In our work, the subject 'what' is composed of one or few input words by users, let us call it Q(what). Then it needs to be identified, parsed and rewritten into the format understandable by LBS, let us call the reformulated what as Q_{what}. We assume a *subject recognizer* has the aforementioned functionality. Figure 7.1 illustrates how the subject recognizer works and how the different components interact:

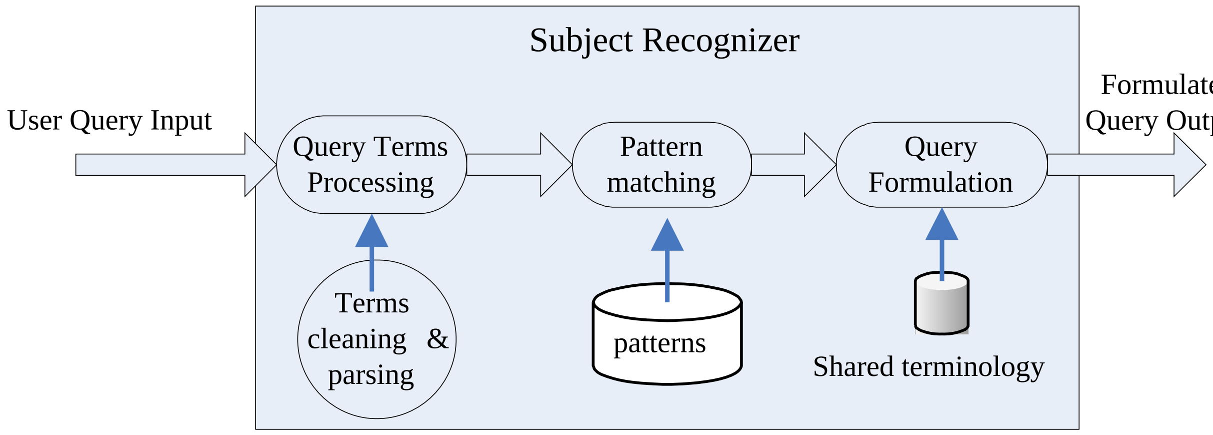


Figure 7.1: The basic infrastructure of the *Subject Recognizer*.

- **Query terms processing**: this process cleans and identifies the terms in Q(what), possibly using assistance from some external dictionary (e.g. WordNet).
- **Pattern matching**: using a set of predefined query patterns, this step identifies the grammatical query pattern of Q(what), e.g. "cheap hotel" is identified as the pattern *adjective + noun*, from which the central term in Q(what), e.g. *hotel* in this example, can be inferred.
- Q_{what} **formulation**: thanks to the Shared Terminology in the LBS, each term in Q(what) is identified and output to the reformulated query, e.g. Q_{what} = hotel, and price="cheap" is added to $Q_{what-else}$.

Informally, the subject recognizer acts to parse and understand what services the user wants to be retrieved by the LBS. By matching Q(what) against the query patterns [AR02], the LBS will understand the subject of user's goal. In other words, the central term and other terms in Q(what) will be identified by looking up the shared terminology. For each pattern, different query rewriting strategies will be applied. Afterwards, by taking into account conditions in other dimensions, as well the user profiles, the LBS can better understand the user's goal and further refine the query. Some basic query patterns using free-text have been elaborated in natural language processing discipline, e.g. [AR02]. According to the actual needs of LBS users, query patterns can be added or modified by LBS experts. Let us now give our strategy in recognizing Q(what) and formulate it to Q_{what} as follows:

Algorithm 1: Reformulation of the what dimension in the Query

```
Input: Q(what) the input as stated by the user; ST is the LBS shared terminology; QP is a set of
       known query patterns.
Output: $q'$, $q'_{other}$ ($q'$ is the reformulated query in the what dimension, $Q_{what}$, and $q'_{other}$ is the
        reformulated query in the other dimensions, if any.
1  begin
2      $q' \longleftarrow \emptyset$; $q'_{other} \longleftarrow \emptyset$;
3      c $\longleftarrow \emptyset$; //initialize a variable in contextualized term data-type;
4      $q = \{q_1, \ldots, q_i\} \longleftarrow \emptyset$; //initialize a variable in string list data-type;
5      $q \longleftarrow Q(what)$;
6      c = PerfectMatch($q$, ST);
7      if $c$ then
8          $q' \longleftarrow$ c;
9      else
10         ($q'$, $q'_{other}$) $\longleftarrow$ TransformQuery($q$, ST, QP);
11 end
```

Algorithm 1 briefly shows how the what dimension in Q(what) is identified and reformulated with the assistance of the service ontology, the shared terminology and the query patterns. The algorithm basically calls two functions: **PerfectMatch**(q, ST) and **TransformQuery**(q, ST, QP). The former checks if there is a perfect match between Q(what) and the name of a service in the service module, either directly or via synonyms from the shared terminology. This obviously is the most favorable case. It is likely to happen when users are looking for the most traditional services, such as hotel, restaurant, bus, museum in a tourism application, or with frequent users, i.e. when users are familiar enough with using the LBS to be aware of the terms used by the LBS. When a perfect match is found, the user input is taken forward to the next processing step, without the need to generate additional specifications at this step. The goal of the second function, **TransformQuery**, is to find out the potential subject of Q(what) whenever PerfectMatch has failed to find the exact match. Once found what type of service the user is looking for, the function further transforms the query into the format (q', q'_{other}). Assume, for instance, Q(what)='experienced baby-sitter' and a perfect match is not found. Lexical analysis of Q(what) shows that it can be matched with the pattern *"adj. + noun"*. At this point the LBS takes 'baby-sitter' as the specification of the targeted service. Let us now assume that there exists a service class called *baby-sitting* in the service ontology. By looking into the shared terminology, the term *baby-sitter* is identified as closely related with baby-sitting: e.g. "baby sitter is a person who can do the work of baby-sitting". Thus, the initial query "baby-sitter" can be rewritten into *baby-sitting*, which becomes the output *q'* (i.e. the subject of Q(what)). The remaining adjective part 'experienced' is then identified as q'_{other}. To better formulate q'_{other} = 'experienced' in terms of LBS semantics, it is necessary to look up the metadata of service class 'baby-sitting' in the service ontology. For instance, if there is an attribute *skillLevel* with value domain {"beginner", "experienced"}, q'_{other} can be formulated as q'_{other}: skillLevel="experienced". The algorithm below details our strategy in reformulating queries when the perfect match can not be achieved, namely the function **TransformQuery**(q, ST, QP), as follows:

Algorithm 2: Transform the Initial Query q into format (q', q'_{other})

```
Input: q is the query Q(what) in string data-type; ST is the shared terminology in LBS; QP is a set
       of query patterns.
Output: $q'$, $q'_{other}$ (The query is transformed into elements understood by the LBS core ontology).
1  begin
2      $q' \longleftarrow \emptyset$ $q'_{other} \longleftarrow \emptyset$;
3      c ⟵ ∅; //initialize a variable in contextualized term data-type;
4      $q_s, q_o \longleftarrow$ IdentifyPattern($q$, QP); //identifies the pattern that corresponds to Q(what) and
       extracts from the pattern the subject of the query returned as $q_s$ and other query specifications
       returned as $q_o$;
5      c = getSimilarService($q_s$, ST); //identifies the service in the service ontology that best matches
       similarity with $q_s$;
6      switch c do
7          case hasSubtype(c, c1) and isSimilar(Discriminator(c, c1), $q_o$)
8              $q' \longleftarrow$ c1;
9          //if the identified service c has a more specific service c1 whose similarity with $q_s$ also
           encompasses the elements in $q_o$, than replace the service c with its more specific subtype c1;
10         case hasProperty(c, p) and hasValue(p, $q_o$)
11             $q' \longleftarrow$c, $q'_{other} \longleftarrow$(p = $q_o$);
12         //if $q_o$ is the valueof a property of c, $q_o$ is interpreted as a predicate on this property and the
           predicate is added to $q'_{other}$ while c is confirmed as the subject of the query and is returned as
           $q'$;
13         case hasSubtype(c, c1) and hasFunctionality(c1, f) and isSubtype($q_o$, f)
14             $q' \longleftarrow$ c1, $q'_{other} \longleftarrow$ (f = $q_o$);
15         //if $q_o$ is a supertype of the functionality of a subtype c1 of c, $q_o$ is interpreted as a predicate
           on the functionality of c1 and the predicate is added to $q'_{other}$ while more specific c1 is stated as
           the subject of the query and is returned as $q'$;
16         case . . .
17             . . . ;
18         otherwise
19             RETURN;
20 end
```

Users are often unable to specify their query appropriately, i.e. in a way that naturally matches the meta-data in the core ontology. This poses a great challenge to the LBS to correctly understand and reformulate the user's query according to user's implicit input. In [AR02], Allan et. al. proposed to apply query patterns to identify user's query. On the basis of observation and analysis on user's queries, they identified a set of frequently used query patterns that can be useful to disambiguate the user's query. Following this idea, we assume there are a set of query patterns known to the LBS, so that for instance a Q(what) formulated as 'experienced baby-sitter' can be found to be consistent with the pattern 'adjective + service'. Accordingly, for each pattern, certain query transformation strategies will be applied. For instance, for the pattern 'adjective+ service', service in the pattern will be identified as the query's subject and adjective will be regarded as other conditions as shown in line 10 of Algorithm 2. In addition, with the assistance of the shared terminology (or external ontology), the query may be identified as a specialization/synonym of existing services in core ontology. For instance, let us assume that Q(what)='ski rental', and in the CO service module there is a service hierarchy showing SportEquipRenting as a subtype of a more generic Renting service: Renting $\sqsupseteq$ SportEquipRenting. Applying the functions **IdentifyPattern**() and **getSimilarService**(), 'rental' is identified as the subject of the query, 'ski' is set as additional specification, and Renting is identified in the service module as the service most similar 'rental'. Rather than stopping here, the algorithm now checks if it can make use of the additional specification 'ski'. It finds 'ski' in the ontology is a sport equipment, $\mathsf{Ski} \subset \mathsf{SportEquip}$, and sport equipment is used to create a specialized subtype of Renting, called SportEquipRenting: $\mathsf{SportEquipRenting} \equiv \mathsf{Renting} \sqcap \forall \mathsf{hasProduct.SportEquip}$. Consequently, rather than generating a generic query for service Renting with additional specification 'ski', a more focused query for service SportEquipRenting can be directly generated. According to the rule in line 11 of the allgorithm, the query is transformed into (q'=SportEquipRenting, $q'_{other} \longleftarrow (\exists \mathsf{hasProduct.Ski})$.

Algorithm 2 as shown here only lists transformation rules for some typical query patterns (lines 7, 10, and 13). The real algorithm would obviously cover all the known patterns and have a case for each one. Once the LBS is in use, new query pattern and transformation strategies can be added and existing ones updated. Moreover, to simplify the algorithm, we did not take into account multiple conditions in Q(what). For instance, the specification *cheap calm hotel* can be regarded as hotel with two restrictions, separately "cheap" and "calm". In this case, by using a loop, all conditions in q_o can be similarly transformed into the format in Algorithm 2, and the final q'_{other} be expressed as a conjunction of concept restrictions. Finally, if the LBS can not identify the service and its subject, its failure means that the current LBS has no service information concerning the service requested by the user. A possibility is then to forward the query to neighboring LBSs.

7.3.3 Spatial dimension - where

The spatial dimension specification is intended to constrain the geographical location or range of the services targeted by LBS queries. As discussed in Chapter 2, relying on loosely-coupling with GIS or spatially-extended DBMS (SDBMS), the LBS can separate spatial data management from thematic data management (i.e. LBS ontology). For each service in the LBS, its geometry data is stored and manipulated via the GIS/SDBMS, but its non-spatial data is organized and handled by the LBS and its data sources (see Definition 4.10 in Chapter 4). In this section, we firstly discuss what spatial operators and functions are mandatory or user-preferred in LBS interactions, then we investigate how to identify semantics of the arguments in Q(where), and finally explain how to translate Q(where) into Q_{where} understood by the LBS.

Spatial Operators in LBS query. Languages for querying geographical databases exist and often call for sophisticated spatial analysis functions that need to be elaborated by experts in geodata manipulation. Non-expert and occasional users are supported via simpler, yet not obvious interfaces in extended SQL (e.g. TSQL2) or alike. These also support map-based queries that refer to a point designated by the user on a map displayed on the screen, LBS cannot offer the same functionality as these interfaces. They can only rely on simple means to specify how to use spatial features in the selection of services. In daily life, most people are accustomed to describe their position (or the position of an object) by referring to a certain landmark (e.g. a hotel, a park, a main street) and using some locational natural language qualifiers such as near, at the right-side, behind, between, etc. These qualifiers informally express a spatial constraint, which can be more formally reformulated as (spatial relationship, spatial argument(s)).

Relationships on spatial objects are generally classified into five types: set-oriented, metric, directional, topological and Euclidean [SC03]. The most intuitive and hence most frequently used in LBS querying are the metric, directional and topological relationships. A recent document from the OGC Consortium (release No. OGC 05-128) identifies eight types of topological relationships: Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains and Overlaps. The metric relationships mainly include Within_Of and the fuzzy relationship Near. Directional relationships encompass three categories [LSC03]: 1) absolute directional relationship that refer to a global reference system, e.g. to the south of, 2) object-based directional relationship, where the direction is based on the orientation of a reference object, e.g. in front of the Opera House, and 3) view-based directional relationship, where the reference is relative to the individual looking at the scene. As our is LBS querying, we focus on providing spatial constraints for service matching rather than navigation or routing functionality. Hence, we mainly investigate the application of metric relationships, some topological relationships, and simple directional relationships for the specification of spatial criteria in LBS queries. Table 3.2 gives a classification of these spatial relationships that are potentially useful in LBS interactions. Referring to Oracle spatial 10g, we illustrate how to represent these spatial relationships in a high-level querying language. Oracle spatial 10g has extended its expressiveness and efficiency to topological and metric relationships, but is still limited as far as support of directional relationships is concerned. In existing GIS or spatially-enabled applications, advanced functions such as directional queries can be realized by programming with e.g. Java or C++ [LSC03].

Spatial Arguments in LBS query. As discussed at the beginning of this section, the spatial argument in the LBS query is expressed in natural language and mainly refers to the spatial reference identifiable or known by common users, which encompasses landmarks, administrative references (e.g. district names), and terms in transportation networks. Benefits of this approach include: 1) ease to recognize, remember and apply by common users in an unfamiliar environment; 2) ease for LBS designers to map between GIS and spatial concepts in the LBS geographical ontology; and 3) ease to extend into other functions, e.g. a map-viewer. Let us firstly investigate the characteristics of the spatial arguments in LBS and then discuss how to disambiguate their spatial semantics.

Locally Constrained. By definition of LBS, the spatial arguments only target services within the local region. Hence, the set of frequently-used references can be narrowed down within the spatial coverage of the LBS, which makes it possible to contain and organize all references in the LBS ontology according to their spatial relationships and geometric features (see Chapter 3).

Imprecision. The imprecision of spatial arguments can result from various reasons, such as the imprecision of positioning, spelling mistakes, the use of indirect references (e.g. Lausanne's oldest church), and multiple names in the geographical ontology referring to a single place in the GIS. Our work mainly focuses on the last two types of imprecision: indirect references and multiple occurrences.

Incompleteness. In database systems, each argument is well defined and is associated with a certain data type and class (or attribute) so that it has no or little ambiguity in the argument. But for location-based services, due for example to the frequent use of abbreviations, the argument in the query can be uncertain and incomplete. For instance, the argument 'St. François' is vague for the LBS since it can correspond to multiple occurrences, e.g. a square, a church, a caf, or a museum. Such ambiguity is a big challenge in spatial web information retrieval. For instance, given a keywords-based query 'Chalet near Lausanne', plenty of results are irrelevant as the search engine cannot identify whether the query looks for places near Lausanne or place names including keywords Lausanne and Chalet.

Context-dependency. The semantics of a spatial argument is often concerned with a certain context, e.g. spatio-temporal one. In particular, the semantics of fuzzy concepts may be dependent on the query's context. For instance, how many kilometers does 'far' mean? In Beijing and in Lausanne, 'far' may respectively stand for 50km and 10km, considering the difference of city areas. The available transport means also have an effect on the argument disambiguation: for instance, 'far' may mean 5km for a pedestrian but ten times longer for a user driving a car.

By default, Q(where) is assumed to use the spatial relationship near, and its a single argument is assumed to represent *Me* (i.e. user's current location). For more expressive spatial constraints, the spatial arguments are not necessarily limited to 'current location'. They can also use other types of references, e.g. street name, district name, or a distance etc. In the former case, the LBS needs to geo-code the user's current position obtained from a GPS into its spatial corresponding position in the GIS, e.g. address; but in other cases, the LBS may need to carry on more complicated tasks to match the arguments against the spatial references in the geographical ontology. In the geographical ontology, any spatial reference has two intrinsic properties: geometry type and reference type. Geometry type corresponds to the geometry type of the spatial reference in GIS, e.g. point/polyline/polygon. Reference type refers to the function type of the spatial reference, e.g. shop, park, highway, town hall etc.

Algorithm 3: Identify the spatial argument(s) in Q(where)

```
Input: Geo_onto: spatial concepts in core ontology; arg: the spatial argument in Q(where); uID:
       user's identifier to position the user.
Output: s_arg[ ]: a set of possible spatial arguments identified by LBS.
1  begin
2      s_arg[ ] ⟵ ∅; //initialize a variable list in spatial data-type of LBS;
3      switch Type(arg) do
4          case NULL
5              RETURN;
6          case "Me"
7              s_arg[ ] ⟵("loc", getLocation(uID)); RETURN ;
8          //the spatial argument is just the location of the user. This is denoted by using keyword "loc"
           as the type of the argument and using as argument value the value returned by the
           getLocation() function. The function returns the geographical position of this user;
9          case address, street, zipCode, districtNo, ...
10             s_arg[ ] ⟵ (Type, arg); RETURN ;
11         //the spatial constraint is explicitly specified in the query. The spatial argument is set to the
           value given in the query, with type denoting the corresponding property (either address or
           street or ...).;
12         otherwise
13             . . .
14     for x ∈ Geo_onto do
15         if MatchLandmarkName(x, arg) then
16             Append(x, s_arg[ ]) //this loop is to convert all landmarks mentioned in the query into
               references to the corresponding ontology element;
17 end
```

Algorithm 3 shows how the arguments are identified in Q(where). When the argument is "Me" (i.e. the user's current location), the LBS gets the position of the user (by whatever positioning system it uses) and provides this position to the GIS as the identified argument (line 7). Similarly, if the type of argument is explicitly identified as a valid administrative spatial reference, such as address, street, zipcode or district number, the argument is accordingly rewritten, e.g. *arg ⟵ (address = "Av. du Leman 23, Lausanne")* (line 9). As we explained before, the references are locally constrained, thus, the city "Lausanne" is added to the new rewritten argument considering user's current location. The function **MatchLandmarkName**() compares the argument with the landmark references in the core ontology. The comparison is based on the words in arg. For instance, if arg = "St. François", the LBS will find three relevant entries in the core ontology, namely Church St. François, Palace 'St. François' and Street 'Av. St. François'. In this case, the LBS will consider the three landmarks as candidates for addition to the argument set. The candidate landmarks are checked for consistency with the spatial relationship in Q(where) (see Algorithm 4 hereinafter). For instance, relationships such as behind or in front of refer to a point or polygon object, not to a linear object. Thus, if Q(where) is "behind St. François", the interpretation of "St. François" as a street can be discarded. Conversely, relationships such as along imply a reference to a polyline or polygon object (e.g. a street or the Leman lake), leading to discard point objects (e.g. shops). Disambiguation between remaining candidate arguments can be performed using additional information, for instance context factors. The most suitable argument will be chosen as the final argument.

Algorithm 4: Transform Q(where) into q_{where}

```
Input: Geo_onto: spatial concepts in the core ontology; arg[ ]: the spatial argument(s) in Q(where);
       rel: spatial relationship in Q(where); uID: user's identifier to position the user.
Output: $q_{where}$: the reformulated query in the spatial dimension.
1  begin
2      n= getArgNumber(arg[ ]);
3      if n=0 then
4          RETURN
5      if n=1 then
6          foreach $x \in arg$[ ] do
7              if CheckConsistency(rel, x) then
8                  $q_{where} \longleftarrow$ Append(rel, x)
9              //rel represents the spatial operator specified in the original query Q(where); x is the
               unique spatial argument in Q(what). This function checks the syntax consistency between
               rel and x, e.g. to check if the spatial data-type of x is consistent with rel. It will return a
               boolean value, if it is "True", append (rel, x) to reformulated query $q_{where}$; otherwise,
               continue to check the next argument in the list arg[].
10     if n=2 then
11         Arg1[ ] $\longleftarrow$ GetArgs(1, arg[ ]);
12         //set all possible first arguments to the array Arg1[ ].
13         Arg2[ ] $\longleftarrow$ GetArgs(2, arg[ ]);
14         //set all possible second arguments to the array Arg2[ ].
15         foreach $x \in Arg1$[ ] do
16             for $y \in Arg2$[ ] do
17                 if CheckConsistency(rel, x1, x2) then
18                     $q_{where} \longleftarrow$ Append(rel, x1, x2)
19             //For any pair of (rel, x1, x2), check its syntax consistency. The consistency check includes
               the consistency of data-types of two arguments, the consistency between arguments and the
               operator rel.
20     $q_{where} \longleftarrow$ ContexualizeQwhere($q_{where}$, getLocation(uID), $context\_factors$)
21     //This function refines $q_{where}$ by considering contextual factors and outputs the most suitable
       spatial predicate $q_{where}$. The refinement mainly refers to specifying relevant parameters, e.g.
       Near(x) can be transformed to Inside(x+$\theta$), the value of $\theta$ can be different according to the user's
       transport means.
22 end
```

The misconception or ambiguity is a common problem in query formulation, in particular when we allow spatial landmarks as arguments. Therefore, Algorithm 4 attempts to check the integrity consistency between spatial arguments and the spatial operator, to eventually better transform Q(where) to q_{where}. The function **CheckConsistency**() checks if the spatial argument is valid for a certain spatial relationship. For instance, the relationship *along* must have an argument in the Line/PolyLine data-type. Thus, if the argument is a spatial reference in point data-type, it is inconsistent with the relationship specification and has to be removed from the list.

The function **ContexualizeQwhere**() refines q_{where} by considering the effect of relevant contextual factors on spatial constraints. For instance, when the user is driving a car, θ 's value of the constraint In(x+θ) is different from when the user is walking. In addition, it may help to eliminate some irrational arguments from q_{where}, e.g. when the user is driving on the highway, for constraint along(highway A2), LBS will only consider the services ahead rather than services behind. In addition, the LBS may need to evaluate and rank the formulated Q(where) by considering other dimensions, for instance, the time dimension. When a user is looking for a service for tomorrow (or in some future time), the current highway constraint is not any more to be considered as a constraint for reformulating Q(where). The inter-dimensional constraints will be checked when processing the query, which will be discussed in the upcoming chapter.

7.3.4 Temporal dimension - **when**

In daily communication, when a user looks for a service, her query is likely not to mention a temporal constraint. This is because the temporal constraint is inferable from local conventions ruling the requested service (e.g. given a query about shops, shops' opening-hours are the same for most shops and can be found in the context data), or because the service is offered anytime (e.g. download of music). However, some types of services may have temporal availability different from the conventional ones, e.g. cinema services. Moreover, some may regularly provide prolonged or reduced services for certain periods in each calendar year, e.g. bus services during summer vacation. Hence, expressing a temporal constraint in Q(when) enables LBS users to specify when they want to have access to a service, without a priori knowledge of the temporal characteristics of the targeted services. Notice that the temporal input Q(when) may determine the spatial characteristics of a service, i.e. the spatial characteristics may be temporal-dependent, e.g. reduced coverage of bus-service on Saturday evening. Therefore, evaluating the Q(when) specification may also be important for evaluating the Q(where) specification.

Temporal relationships in LBS. In last two decades, the temporal database community produced many proposals for temporal data models, which aimed at improving the current status of temporal data management [PSZ06] [JS99] [TCG$^+$93]. Logic-oriented research mainly focuses on the description of the underlying inference system of temporal expressions. Regarding temporal relationships, although the temporal operations and data types in each data model can be slightly different, the commonly-used temporal predicates are generally based on the specification of Allen's algebra of temporal interval relationships [All83]. However, in daily communication, people use slightly different temporal prepositions and subordinating conjunctions to specify temporal relationships. However, temporal database systems only provide little support for representing these user-preferred temporal descriptions. Table 7.3.4 lists the preposition-alike temporal relationships that we propose to support within the LBS, aiming at simplifying the use and identification of temporal relationships by casual users.

Defining the semantics of these user-preferred temporal prepositions is not straightforward. Specific problems arise. For instance, the same preposition may be used with more than one semantics, e.g. IN five minutes, IN winter. Often, more than one preposition may be used to give the same or very similar meanings, e.g. IN winter and DURING winter. These ambiguities and overlapping meanings make it difficult for computer systems to deal with. In our LBS temporal interactions we cope with the problem by specifying a limited number of NL-based prepositions, with a specific semantics, to describe the temporal relationships between the availability of services and a given timeframe (the timeframe of interest to the user).

Table 7.2: Examples of temporal queries in LBS.

Temporal relationships in the LBS interface	Semantics of the temporal relationships	ServiceTime x in temporal query	Temporal matching between query and service	Corresponding argument(s)
At(t)	Return services available exactly at an instant t C_s	t ⊆ x	Equal(ServiceTime, x) or Covers(ServiceTime, x)	t: current or future instant, e.g. 3pm, noon
On(t)	Return services available at day of week t	x = t	Covers(ServiceTime, x)	t: upcoming day of week, e.g. Friday.
Around(t)	Return services available at approximately instant t	x ∩ duration(t-△, t+△)≠ ∅	Overlaps(ServiceTime, x)	t: current or future instant, △ is a small variant defined by LBS
Before(t)	Return services available before instant or interval t	x ∩ duration(Now, t) ≠ ∅	Overlaps(ServiceTime, x)	t : current or future instant (e.g. 5:00pm) or interval (e.g. lunch)
After(t)	Return services available after instant or interval t	x ∩ duration(t, t+△) ≠ ∅	Overlaps(ServiceTime, x)	t : current or future instant (e.g. 5:00pm) or interval (e.g. lunch), △ is variant defined by LBS
During(t), In(t)	Return services overlapping interval t	x ∩ t ≠ ∅	Overlaps(ServiceTime, x)	t: future interval, e.g. this afternoon
From(t1) to(t2)	Return services covering the interval from instant t1 to instant t2	x = duration(t1,t2)	During(x, ServiceTime)	t1: a current or future instant, t2: a future instant, and t1<t2
Every(t)	Return services always available at a certain instant/interval	x[i] =InstantBag getInstants(t) or x[i] =Intervalbag getIntervals(t)	During(x[i], ServiceTime), for all i.	t : a fixed instant or interval, e.g. Monday afternoon

In temporal databases, temporal methods and operations are directly performed on two explicit time instants or intervals, e.g. to assess if a date is before another date. In the LBS interface, the user specifies the desired timeframe by choosing the temporal relationship and specifying the corresponding argument(s). From this input, the LBS first validates the relationship and its argument(s), and then translates it into a simple or complex time explicit specification in a format suitable for further processing by the LBS. The reformulated temporal constraint is eventually matched against the temporal availability property, **ServiceTime**, of the candidate services the in LBS ontology.

Temporal arguments in LBS. The LBS ontology provides concepts for the description of the temporal characteristics of services, as discussed in chapter 3. These descriptions include the temporal data types and values normally using the Gregorian calendar. However, considering the contextual framework, e.g. local culture and tradition, the LBS ontology may also include a local calendar, e.g. a Chinese Lunar calendar. In addition, the LBS ontology may include some commonly-used temporal terms: 1) explicit instants or intervals such as Now, Noon, Today, Tomorrow, Weekend, 2) fuzzy temporal intervals such as afternoon, morning, evening, soon, 3) context-dependent intervals, e.g. lunch time, Chinese new year, and 4) Sequential terms such as *this, next*.

Regarding temporal granularity, i.e. which unit is chosen as granule (the smallest temporal value), an LBS will usually support an enumerated set of possible granules, year, month, day, hour, minute, compatible with a Gregorian calendar. Although finer granularity (second, millisecond) are theoretically available, from the practical viewpoint there is no need for LBS to get to that level of accuracy as it is irrelevant to human interaction.

Characteristics of Temporal Query Formulation.

1. *Services in current or future time.* Different from the temporal queries in databases, the service requests to LBS only apply for querying in current or future time. Hence, the temporal relationship before

implicitly refers to an interval that starts at instant Now and ends at the instant variable t in the argument of before.

2. *Have an implicit lifecycle.* Temporal constraints specified in the LBS interface often refer to an implicit short period, e.g. today, this week. For instance, if the user specifies after(5:00pm), it implicitly denotes the closed interval 'from today 5:00pm until today 12:00pm'.

3. *Fuzzy temporal operators.* In Table 7.3.4, we have only one fuzzy relationship, around (temporally near an instant), which is common in daily communication where great accuracy is usually not mandatory, or the exact temporal property of a service may be unknown, e.g. when the next bus is coming. The associated variable $\triangle$ can have different temporal semantics, i.e. granularity and value, it mainly depends on the temporal semantic of the temporal argument t. For instance, around(lunch time), the granularity and the value of $\triangle$ may be respectively 'hour', and 1 or 0.5, since the lunch-time is about one hour. A similar example, frequent in current web services, is the booking of a flight around a certain day, where $\triangle$ has options such as 1day, 3days.

4. *Regular temporal relationships.* Sometimes the user may look for a service offered regularly at a fixed instant/interval in a longer period (e.g. a month or a semester), for instance, 'a Latin-dance course every Tuesday evening'. Hence, this type of complex temporal relationship often associates with a period which need to be explicitly expressed, e.g. in this semester, from January to March. It can be implemented by combining more than one relationship: firstly getting all instants/intervals within the lifecycle according to the query, and then testing whether the services' temporal availability covers all of them.

5. *Context influence.* In the discussion above, we ignore the fact that it takes some time for users to get access to the service if the user asks for a service in near future, e.g. in two hours. In reality, many contextual factors, such as traffic and spatial constraints, should be taken into account in order to access to services on time.

6. *Complex temporal relationships.* Sometimes, single temporal relationships cannot sufficiently express the desired temporal constraint. The use of Boolean operators between two time instances, such as OR (e.g. in the weekday afternoon or Saturday morning), AND (every Saturday AND from January to March) helps addressing these cases. Complex relationships may lead to more validation work to check consistency between the two time instances.

Similar to the strategy for query reformulation in the spatial dimension, query reformulation in the temporal dimension focuses on three aspects: 1) identify the temporal argument(s), 2) validate the consistency between the temporal arguments and relationships in Q(when), and 3) reformulate Q(when) into q_{when} that can be understood by LBS. Algorithm 5 describes how to reformulate the query in the temporal dimension.

When reformulating the query in the temporal dimension, we assume we have an embedded calendar/clock that can accurately identify the local time of the current user. In the function **MatchTemporalArg**(), the argument will be matched with the concepts in Time_onto (i.e. temporal concepts in the core ontology) considering the current time. For instance, when the argument is *afternoon*, the LBS can encode the time interval of afternoon in terms of temporal expressions understood by LBS. Today/tomorrow can be transformed into the date format which can be computed by the LBS. Another function **CheckConsistency**() validates the inputs in Q(when), e.g. it can not refer to past time, and checks the consistency between them, i.e. consistency between the relationship and the arguments, as well as consistency between arguments. For instance, if the relationship is From t1 to t2, t1<t2 must hold. In addition, complex temporal relationships can be expressed by combining two simple temporal relationships. For instance, every Saturday afternoon from January to June can be enumerated as a set of time intervals by referring to the calendar.

Algorithm 5: Transform Q(when) to q_{when}

```
Input: Time_onto: temporal concepts in the core ontology; arg[ ]: the temporal argument(s) in
       Q(when); rel: temporal relationship in Q(when).
Output: $q_{when}$: the reformulated query in the temporal dimension.
1  begin
2      n= getArgNumber(arg[ ]);
3      //get the number of temporal arguments in the query.
4      if n=0 then
5          $q_{when} \longleftarrow \emptyset$;
6          RETURN.
7      if n=1 then
8          x1 = MatchTemporalArg(arg[1], Time_onto);
9          if CheckConsistency(rel, x1, Now) then
10             $q_{when} \longleftarrow \mathsf{Append}(rel, x1)$;
11         //The function MatchTemporalArg() firstly validates the input argument x1 by comparing
           current time NOW with x1, then checks the temporal consistency between rel and x1. If it
           returns True, (rel, x1) will be output as the reformulated $q_{when}$.
12     if n=2 then
13         x1 = MatchTemporalArg(arg[1], Time_onto);
14         x2 = MatchTemporalArg(arg[2], Time_onto);
15         if CheckConsistency(rel, x1, x2, Now) then
16             $q_{when} \longleftarrow \mathsf{Append}(rel, x1, x2)$;
17         //Beyond the argument validity (with NOW) described above, this function needs to check the
           consistency between two arguments, e.g. From(x1)To(x2), x1, it must hold that x1 is earlier
           than x2.
18     $q_{when} \longleftarrow$ ContexualizeQwhen($q_{when}$, NOW, $context\_factors$)
19     //This function refines $q_{when}$ by considering contextual factors and outputs the most suitable
       temporal predicate $q_{when}$. The refinement mainly refers to specifying relevant parameters, e.g.
       Around(x) can be transformed to Between(x-$\theta$)AND(x+$\theta$), the value of $\theta$ may be different
       according to the local convention or user's current activity.
20 end
```

7.3.5 Other functional conditions - What-else

Besides the subject constraint, spatial and temporal constraints, the user can specify other general conditions which span over any properties of the service, e.g. their offered functionality, their title, their quality of service, etc. In the interface, we allow user to specify conditions using keywords or predicates in the format of (property, operator, variable), separated by commas. If more than one item is input in 'what-else', the order of items is possibly interpreted as an order of priorities. The keywords are matched with the descriptions (or annotations) or properties of the corresponding classes in the ontology or the data sources. The algorithm for reformulating the query in the What-else dimension is described in Algorithm 6. It firstly identifies each condition in Q(what-else). We assume each tuple can be transformed to the format (property, operator, variable) by considering the service description in the core ontology. For instance, price $<$ 30CHF can be easily transformed to (price, $<$, 30), keywords "historical" for a movie service can be transformed to a tuple like (type, include, "historical") or (description, contain, "historical") where *type* and *description* can be descriptive properties of the service *movie*. In addition, the contextual factors may play an important role in refining q_{else}, e.g. *cheap* can be transformed into a tuple such as ($\leq$, price, "*low*"). The concrete value of "low" in this example can be determined by multiple factors, e.g. the local market. Finally, the function **CheckConsistency** checks the consistency between the conditions in Q(what-else), e.g. ($>$, price, 100) conflicts with ($<$, price, 90).

Algorithm 6: Transform Q(what-else) to q_{else}

```
Input: CO: core ontology; $q_{what}$: reformulated query in what dimension; arg[ ]: a set of condition
       tuples in Q(what-else).
Output: $q_{else}$: the reformulated query in what-else dimension.
1  begin
2      $q_{else} \longleftarrow \emptyset$;
3      // initialize $q_{else}$ as empty.
4      foreach $t(x, op, v) \in arg[\,]$ do
5          //parse each tuple t in Q(what-else), x: property, op: operator, v: variable. e.g. (price, <, 30).
6          p = $\emptyset$ ;
7          p = TransformTuple($t, q_{what}, CO$);
8          //Refering to the corresponding service's metadata in CO, this function firstly identifies x with
           the property of service class $q_{what}$, then transform t it to p which can be further processed by
           LBS. This identification process is similar to mapping a property of the service profile to a
           property of service class in CO.
9          if CheckConsistency(p, $q_{what}$, $q_{else}$) then
10             $q_{else} \longleftarrow$ Append(p)
11 end
```

7.3.6 Query Formulation: a Complete View

In this last section we illustrate on an example how a query is understood and reformulated in all dimensions. However, conditions in the diverse dimensions may interact and have an influence onto each other. When we combine these conditions in a single one, we need to check the dependencies and conflicts, and further decide how to modify and reformulate them.

Example 7.1. Given the original query Q(what, where, when, what-else) := (Salsa dance course, near Malley, on Thursday evening, 'beginner'), we show how to transform it into a conjunctive query.

Algorithm 7: Transform the whole query Q to q_{all}

```
Input: CO: the core ontology; $q_{what}$, $q_{other}$, $q_{where}$, $q_{when}$, $q_{else}$.
Output: $q_{all}$: the complete reformulated query.
1  begin
2      ConflictCheck($q_{what}, q_{other}, q_{where}, q_{when}, q_{else}$);
3      $q_{all} \longleftarrow (q_{what}, q_{other}, q_{where}, q_{when}, q_{else})$;
4  end
```

Step 1: Q(what) Reformulation. We assume the LBS ontology contains a service concept 'DanceCourse' and its super-concept is 'Course', but does not contain the concept 'SalsaDanceCourse' or another similar concept. With Algorithm 1 and Algorithm 2, the original query Q can be rewritten as:

q_{what}= "DanceCourse" and q_{other}= (style = 'Salsa')

$\Rightarrow$ (x) $\longleftarrow$ DanceCourse(x) $\wedge$ Style(x, Salsa)

Step 2: Q(where) Reformulation. In Q(where), the user specifies the *near* relationship and its argument as "Malley". In the core ontology, Malley is identified as an administrative district in Lausanne. Hence, the Q(where) is reformulated into:

q_{where} = near(district, Malley)

$\Rightarrow$ q_{where} = Distance(Malley.location, service.location) = n;

(informatively formulated as above, where n is the distance between two locations.)

$\Rightarrow$ q_{where} = SDO_NN (Malley.location, service.location, 'SDO_NUM_RES=1km') = 'True';

(formulated in Oracle with context-dependent value n = *1km*)

Step 3: Q(when) Reformulation. The temporal constraint is interpreted as implicitly meaning the upcoming Thursday evening. By referring to the current date, the upcoming Thursday can be calculated. Let us assume current date is "2007-07-17, Tuesday", in this context, *Thursday* here means "2007-07-29, Thursday".

q_{when} = on ("Thursday evening")

$\Rightarrow$ q_{when} = BETWEEN["2007-07-19,19:00","2007-07-19,24:00"]

$\Rightarrow$ (x) $\longleftarrow$ DanceCourse(x) $\wedge$ OpenTime(x,y) $\wedge$ BeginDate(y, 2007-July-19) $\wedge$ BeginHour(y, 19)$\wedge$ EndDate(y, 2007-July-19) $\wedge$ EndHour(y, 24)

Step 4: Q(what-else) Reformulation. The initial conditions in Q(what-else) is only one keyword 'Beginner'. By identifying its value, it is found it matches with the value domain of attribute level, hence it is reformulated as follows:

q_{else} = (level = 'beginner')

$\Rightarrow$ (x) $\longleftarrow$ DanceCourse(x) $\wedge$ Level(x, Beginner)

Step 5: q_{all} Formulation. After conflict checking, there is no condition conflict between different dimensions. The final query q can be written in the following way:

(x)$\longleftarrow$ DanceCourse(x) $\wedge$ Style(x, Salsa)$\wedge$ Level(x, Beginning) $\wedge$ OpenTime(x,y) $\wedge$ BeginDate(y, 2007-July-19) $\wedge$ BeginHour(y, 19) $\wedge$ EndDate(y, 2007-July-19) $\wedge$ EndHour(y, 24)

7.4 Chapter Summary

This chapter is devoted to explaining how the LBS performs the very first task in query processing, which we call query reformulation. After an introductory overview of various query formulation approaches, not limited to LBS interfaces, we somehow emphasize the rationale behind our assumptions in terms of query formulation. We recall that the concern about supporting a simplest as possible interaction with the user leads us to just ask the user to specify the essential components of her/his request: what service is being requested, where and when it has to be available, and what other conditions shall be satisfied for this particular query. Such a skeletal query, expressed by a user who may be perfectly unfamiliar with the vocabulary used by the LBS to talk about services and the surrounding world, obviously calls for some processing by the LBS oriented towards understanding user's specifications. We sketch a number of algorithms that show the kind of processing performed by the LBS. This includes lexical analysis of the expressions formulated by users (our queries are not restricted to a set of keywords), term matching with the ontological and terminological knowledge stored in the LBS, and checking the consistency of the specifications. This set of techniques eventually leads, whenever possible, to an unambiguous understanding of the user's intention. On this basis the query is reformulated in terms that are acceptable by the next steps in query processing, which are discussed in the next chapter. Given that the focus of this work is on semantic analysis of the issues, not on the implementation of a solution, the algorithms are functionally described but have not been coded and implemented. They have to be understood as a specification of what needs to be done, which is what this thesis aims to achieve.

Chapter 8

Query Processing

8.1 Introduction and Motivation

Query processing is a challenging topic. In relational database systems, *query processing* generally involves two tasks: answering the query and transaction processing. The former one means that the database components (e.g. query compiler) do parse, optimize and execute the query. The latter one refers to the fact that multiple queries may be grouped into transactions, and transactions have to be properly handled so that data's **ACID** properties (where **A** stands for *atomicity*, **I** stands for *isolation*, **C** stands for *consistency* and **D** stands for *durability*) are enforced. SQL (i.e. the standard language for dealing with relational databases) includes both data definition languages and data manipulation languages. They allow definition of the database schema, and manipulation on the schema and on the data in the database.

More modern frameworks such as the semantic web rely on RDF, a basic model for describing web data resources and facilitate their encoding, exchange and reuse. A *resource* is any object which is identifiable by a unique URI (Uniform Resource Identifier). Each resource has a set of properties. Each *property* associates the resource and a *value* (or another *resource*). RDF's data pattern, a triple subject-predicate-object, is rather simple, which facilitates the exchange of diverse application data. A variety of RDF-based query languages have been proposed to support querying information structured as RDF descriptions. SPARQL[1] (W3C candidate recommendation) is based on matching graph patterns and is able to express queries across diverse data sources. In addition, SPARQL introduced binary relations (called *E-entailment regime*) between subsets of RDF graphs to extend the basic graph matching technique. This was also adopted in OWL[2].

In DL-based KnowledgeBases [BLR03], the query can be viewed as a concept description, so that query processing can be regarded as evaluating necessary and sufficient conditions over DL-KnowledgeBase's TBox and ABox. Due to the intrinsic concepts' subsumption hierarchy in DL-KnowledgeBase, queries in DL can be classified, refined and explored within a sub-part of the DL-KnowledgeBase, i.e. the subsumption relationship can be properly applied in query optimization. To overcome the limitations of DL's expressiveness, some researchers proposed to combine DL and Datalog rules to keep the query answering and reasoning decidable,

[1] Refer to the latest specification of SPARQL, http://www.w3.org/TR/rdf-sparql-query/

[2] Refer to RDF Semantics in the W3C community.

see for example the LOGIN and CARIN languages. In addition, to augment the reasoning scalability over large OWL ontologies, some attempts on storing and reasoning ontologies in databases have been made, as shown in LUBM benchmark [GPH05] and OntoMinD [AJPS07].

In LBS setting, the data in each module is provided and maintained by heterogenous information sources. With the assistance of the *shared terminology* and *mapping libraries*, the core ontology holds an integrated view on the metadata of all modules. When a query is issued by a given user, the query is initially formulated in the conjunctive predicate format as described in chapter 7. Next, it is processed by the LBS query processor. There may exist a variety of query languages to retrieve the information from the core ontology. Comprehensive overviews of capabilities and performances of these query languages have been presented in [HBEV04] and W3C website. Current query languages (QLs) for semantic web ontologies can be generally classified into two categories: RDF-based QLs (such as RDQL, SeRQL, and SPARQL etc.) and DL-based QLs (such as DIG ask queries, nRQL), as suggested in [SP07]. The former category is based on matching triple patterns on RDF graphs, but it is difficult to map these triples to well-formed OWL-DL constructs. The latter category has well-grounded semantics based on DL, but it is yet unable to provide sufficient querying functionalities, i.e. make disjunctive and conjunctive queries over the ABox and TBox. Therefore, to define a simple but powerful query language to retrieve and manipulate data in OWL is also a significant open issue to further promote semantic web, as investigated in the recent workshop OWLED'2007 (OWL: Experiences and Directions). Due to the characterization *"mobility, locality and dynamics"* of LBS, the current DL-based querying approach, i.e. to predefine query patterns, is a time-consuming task and hard to implement and maintain in LBS. To simplify the issue, we adopt SPARQL as the query language to briefly explain the (meta)data retrieval process in our work. Query relaxation may be more common in LBS, due to the misconception, incomplete knowledge of local contexts, even the user's desire for additional information. For instance, it may happen when the user can not obtain satisfying results or the LBS fails to find perfect-matching results. Therefore, in our work, *query processing* is mainly concerned about query answering and query relaxation.

This chapter starts with the basic flow of query processing in LBS. Next, we discuss relevant algorithms in query answering, e.g. query refinement with *determine* and *influence* relationships. Then we introduce the basic syntax and semantic extensions of query language SPARQL, and explain how to transform our query in terms of conjunctive predicates into SPARQL-compatible format. By defining the *query relaxation profile*, we show how to define the relaxation rules, and apply them to produce relaxed queries and achieve alternative answers for users. Finally, we discuss future work on how to control relaxations and investigate strategies on multiple relaxations.

8.2 Query Answering

8.2.1 The Basic Workflow of Query Processing in LBS

Figure 8.1 illustrates the basic workflow of LBS query processing. Firstly, with the assistance of shared terminology and core ontology, the user's original query is reformulated into Q in terms of conjunctive

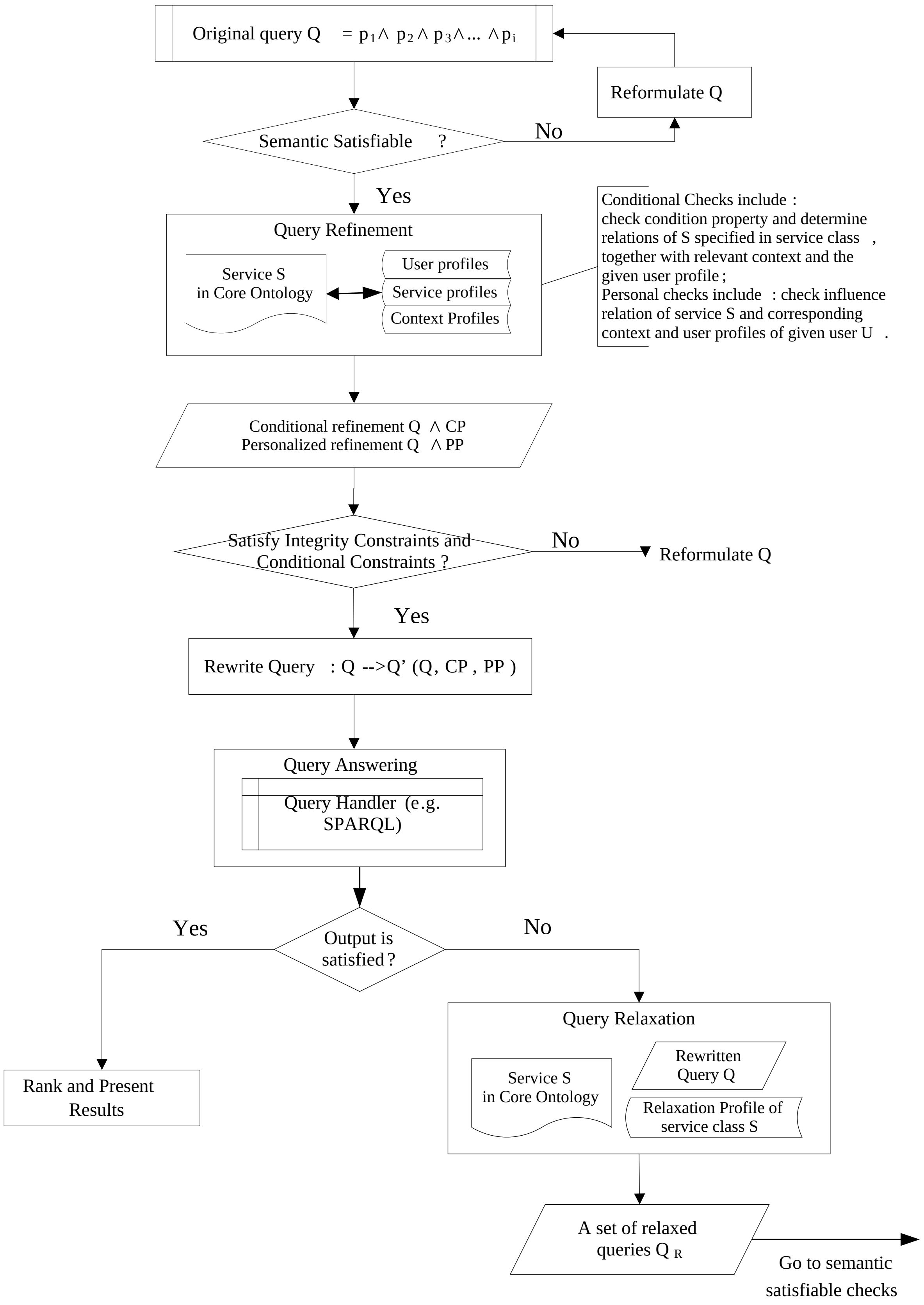

Original query Q = p₁∧ p₂∧ p₃∧... ∧pᵢ
Reformulate Q
Semantic Satisfiable ?
No
Yes
Query Refinement
Service S in Core Ontology
User profiles
Service profiles
Context Profiles
Conditional Checks include: check condition property and determine relations of S specified in service class, together with relevant context and the given user profile; Personal checks include: check influence relation of service S and corresponding context and user profiles of given user U.
Conditional refinement Q ∧ CP
Personalized refinement Q ∧ PP
Satisfy Integrity Constraints and Conditional Constraints ?
No
Reformulate Q
Yes
Rewrite Query : Q -->Q' (Q, CP, PP)
Query Answering
Query Handler (e.g. SPARQL)
Yes
Output is satisfied?
No
Query Relaxation
Rank and Present Results
Service S in Core Ontology
Rewritten Query Q
Relaxation Profile of service class S
A set of relaxed queries Q R
Go to semantic satisfiable checks

Figure 8.1: The basic work-flow of query processing in LBS.

predicates: $Q = p_1 \wedge p_2 \ldots \wedge p_i$ described in Chapter 7. In the query reformulation phase, LBS also check the completeness of the query input. For instance, all *input properties* of a service need to be provided and be able to be identified by LBS. In addition, to ensure the semantic satisfiability of the original query Q, the LBS will validate Q according to certain consistency constraints. We only provide a fundamental guideline about what semantic satisfiability should be taken into account in LBS setting (see details in Algorithm 8), definitely, they can be modified or refined by LBS experts. If the semantics of the query Q is satisfiable, LBS will refine the query Q with the knowledge in the core ontology. On the one hand, the LBS checks if access to the requested service S is subject to pre-conditions (either stated as *precondition property* or as *condition* and *determine* relationships). Any pre-condition has to be analyzed and its satisfiability must be verified with user's information. On the other hand, the user's rules and determine relationships relevant to the service S need to be considered as well. We can consequently refine the original query by adding the conjunction of the set of conditional predicates $CP = cp_1 \wedge cp_2 \wedge \ldots cp_j$ (e.g. cp_i: $\mathsf{User} \sqcap \exists \mathsf{hasMembership}._{=}$\{EuropCar\} for a service that requires the user to have a membership into EuropCar), leading to $Q_{conditional} = Q \wedge CP$. Similarly, the characteristics of target customers of service S, as well as the requesting user's relevant preferences and information for service S (e.g. using *Property Relevance Association*), can be described as disjunctive/conjunctive predicates PP, $PP = (rp_1 \wedge wr_1) \vee (rp_2 \wedge wr_2) \vee \ldots (rp_k \wedge wr_k)$ where rp_x is a relevant predicate for service S for a given user U, and wr_x specifies its importance degree for the service S and the user U (e.g. ∃hasDrivingLicense.$_{=}$\{Swiss B01\} to state that the user has to have a Swiss driving license of type B01), leading to $Q_{relevant} = Q_{conditional} \wedge PP$. $Q_{conditional}$ contains the necessary conditions to service S and user's constraints on service S (i.e. *hard query*), and $Q_{relevant}$ includes the relevant information to help to filter and rank query results (i.e. *soft query*). Afterwards, the semantic satisfiability of $Q_{conditional}$ and $Q_{relevant}$ needs to be validated before the query is syntactically reformulated in SPARQL format, so that it can be forwarded to the query processing engine in charge of executing the service matching and ranking. When a query fails to get answers, the query can be either reformulated by the user, or relaxed by the LBS according to the relaxation profile of service S. The relaxed query will be validated and processed iterating the above steps. We will further discuss query relaxation in Section 8.3.

Algorithm 8 presents how to discover the conditions separately from both services and the user's concerns. The lines 4-7 identify relevant conditions by looking up the user-defined rules in *uID*'s current profile: if any rule involves the service S or its super-classes, then append it to CQ. For instance, assume a rule in a given user profile is "filter out all McDonald's if I am looking for a fast-food shop or restaurant". In this case, when the user asks for a restaurant, the predicate x ∈ Restaurant ⊓ ¬hasName.$_{=}$\{MacDonald's\} is generated. Lines 8-11 take into account the case in which the concept/property-value in the user profile has determining effect on the service S. For instance, the user's driving license directly determines the car types (see Figure 8.2) that the user can rent from Car-Rental services, i.e. $\widehat{r}$(DrivingLicense, Car-rental⊓Car-type), which can be briefly translated in SPARQL triples as follows:

```
Prefix CO
Prefix UP
SELECT ?x
WHERE { ?x rdf:class CO:Car_Rental .
        ?x CO:CarType ?cartype .
        ?u UP:drivingLicense ?dlicense .
        ?dlicense UP:determineCarType ?cartype .
      }
```

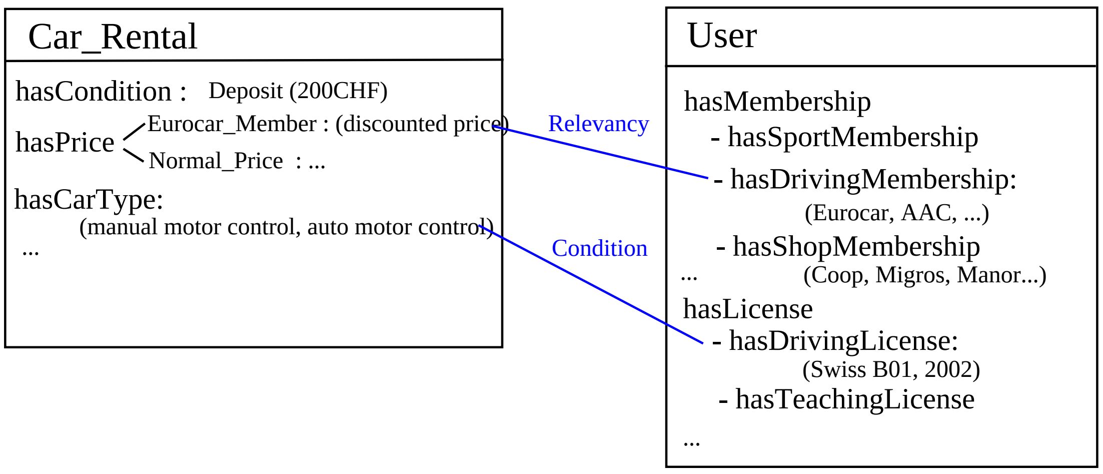


Figure 8.2: The conditions and relevancy between service Car_Rental and a user profile.

Algorithm 8: DiscoverConditions(S, uID): to discover conditions to access service *S* for a given user *uID.*

```
Input: S: the metadata of service class S; UP(uID): the profile of the user uID.
Output: CQ(S, uID): user uID's constraints for service S; CS(S, uID): the service S's conditions for
        the user uID.
1  begin
2      $CQ(S, uID) \longleftarrow \emptyset$;
3      $CS(S, uID) \longleftarrow \emptyset$;
4      for each rule $r \in$ UP(uID) do
5          //The rule r is composed of the head and the body (both consisting of a conjunction of
           RDF-triples, i.e. (a1, p1, b1) $\wedge$ (a2, p2, b2) $\wedge$ ....), denoted as $(h_1, \ldots, h_i) \longleftarrow (b_1, \ldots, b_j)$,
           where each b and h are expressed in the conjunction of triple (x, p, y).
6          if $(h_1, \ldots, h_i)$ is TRUE then
7              for each triple $b \in (b_1, \ldots, b_j)$ where $(b = (S_b, p, x) \wedge (x, op, var))$ do
8                  if $S_b \in S$ OR $S_b \sqsubseteq S$ OR $S \sqsubseteq S_b$ then
9                      $CQ(S, uID) \longleftarrow$ Transform(b, S)
10     //The loop above identifies all rules specified in UP(uID) relevant to service S. If relevancy, it is
       transformed into the predicate understood by LBS then appended to CQ. Rule Relevant to S
       means S' in rule(S') overlaps with S: 1) S' can be an instance of S (S'∈ S), 2) S' can be a supertype
       of S (S⊑S'), 3) S can be a supertype of S' (S'⊑S), e.g. the McDonald's example above.
11     for each property tuple t = (uID, p, x) $\in$ UP(uID) do
12         if $\exists$ property tuple t' = $(s, p_s, x_s) \in S$ AND $\widehat{r}(p, p_s)$ then
13             $CQ(S, uID) \longleftarrow$ Transform(t, S)
14     //This loop identifies all property relevancy associations $\widehat{r}$ between properties of S and properties
       in uID's user profile. If they are relevant, transform the tuple t in UP(uID) to the predicate
       processable by LBS. Please note here relevancy $\widehat{r}$ means determine. For instance, the type of the
       user's driving licence determines the car-type which can be driven by the current user.
15     for each condition relation $\prec$(C, S) on services S do
16         if uID $\in$ C then
17             $CS(S, uID) \longleftarrow$ Transform((uID, type, C), S)
18     //This function checks the condition relation of service S. If the user is in the corresponding User
       Class, then add the predicate (uID, type, C) to the CS. For instance, some services can be
       accessible only if another service has been taken, e.g. the basic ski course is the precondition of the
       advanced ski course.
19     for each precondition property hasPrecondition(S, C) of service class S, where C is a concept which
       can be transformed to a conjunction of triple(s) t=(u, p, x) do
20         if $\exists$ property tuple t' = (uID, p', x') $\in$ UP(uID) AND C $\sqsupseteq$ t' then
21             $CS(S, uID) \longleftarrow$ Transform(t', S)
22     //t = (u, p, x): the property condition t specifies that the user must have the property p with
       value x. It helps to find the corresponding condition properties of service S in uID's user profile.
23     CQ(S, uID) $\longleftarrow$ ConsistencyCheck(CQ[ ]);
24     // to validate the consistency of all conditional predicates from the user uID's concern. The
       consistency checks mainly include: 1) if the necessary information is provided in Q, e.g. the $Q_{what}$
       can not be empty, 2) if the data-type of the variable is consistent with the specification, e.g. the
       location can not be filled with numeric variables, 3) the consistency between predicates holds, e.g.
       the predicate specified in the query can replace the predicate in the user profile.
25     CS(S, uID) $\longleftarrow$ ConsistencyCheck(CS[ ]);
26     // to validate the consistency of all conditional predicates from the service S 's concern, and they
       are similar to the descriptions above.
27 end
```

Similar to a *join* of two relations in SQL, the determinant connection between concepts/properties in different modules needs to be built up using certain constructs or rules. In the above example, both service individual ?x and user's license ?dlicense correspond to the same car type value ?cartype. In other words, the original query Q is reformulated to CQ with the user's conditions specified in the given user profile. Please notice this query reformulation is not same as query relaxation. The lines 12-19 in Algorithm 8 process all conditions to access service S. They can help to construct conditional mappings for service S, i.e. what information of the given user profile is relevant to hard conditions of service S. Finally, it respectively checks the consistency between all conditional predicates in CS and in CQ to ensure the validity of the reformulated conjunctive queries. Moreover, the condition and relevancy between a given service and a given user can be pre-processed as long as the service class and user profile are defined, and they are subject to change while the user modifies the user profile or the service class S evolves, similar to the *view* in databases.

Similarly, the relevance discovery of service S for the given user uID is defined in Algorithm 9. Rather than hard constraints discovery in Algorithm 8, the algorithm below aims at exploring the relevancy between the service S and a given user profile. It allows LBS taking advantage of user profile to personalize query results and to navigate over information in diverse modules. Issues on result ranking and calculation are not our main focus, the discussion on this topic can be found in the work [VHPA06].

Algorithm 9: DiscoverRelevancy(S, uID): to discover relevant information to access service *S* for a given user *uID*.

```
   Input: $S$: the metadata related to service class $S$; UP(uID): the profile of the user $uID$.
   Output: $PP(S, uID)$: the relevant information to select and rank services $S$ for the user $uID$.
1  begin
2      $PP(S, uID) \longleftarrow \emptyset$;
3      for each ordered-value property $p_{\succ}(d, \tau) \in UP(uID)$ do
4          if $\tau \sqcap S \neq \emptyset$ then
5              $PP(S, uID) \longleftarrow$ Transform($p_{\succ}$(d, $\tau$), S)

6      for each property relevance association $\widehat{r}(p1, p2)$ where p1 is a property of service S and p2 is a
       property of UP(uID) do
7          $PP(S, uID) \longleftarrow$ Transform(p2, S)
8      //For a given user profile, the function above discovers all relevant properties in class S from the
       user's viewpoint.
9      for each influence relation $\overrightarrow{i}(C, S)$ for service class $S$ do
10         if $uID \in C$ then
11             $PP(S, uID) \longleftarrow$ Transform((uID, type, C), S)

12     for each preference property $p_{\heartsuit}(t, S)$ of service class $S$, where $t = (u, p, x)$ do
13         if $\exists$ property tuple t' = (uID, p', x') $\in$ UP(uID) AND $\widehat{r}(p, p')$ then
14             $PP(S, uID) \longleftarrow$ Transform(t', S)

15     //It recommends services to certain users who are in the target customer group of the service S.
16     PP(S, uID) $\longleftarrow$ ConsistencyCheck(PP[ ])
17     // check the consistency of all personal predicates and output the result.
18 end
```

8.2.2 Query Translation and Answering in SPARQL

One of the significant issues in current semantic web implementations is the lack of a well-acknowledged and efficient query language, analogous to SQL for databases. Meanwhile, as pioneering efforts to query semi-structured web data, RDF-based query languages attracted much attention from both researchers and practitioners. Consequently, a series of proposals have been made and implemented, e.g. RDQL, SeRQL, and SPARQL. Amongst, SPARQL is a well-developed one and potentially will become the standard RDF-based query language for the semantic web. In this subsection, we will firstly introduce the abstract syntax of SPARQL and an OWL-DL extension SPARQL-DL, then explain how to translate our conjunctive query to the SPARQL-compatible format.

SPARQL is defined on the basis of previous RDF query languages such as rdfDB, RDQL, and SeRQL, and has several valuable new features of its own. SPARQL can make simple data retrieval on RDF graphs by restricting string or numeric values on subject, predicate and object. Further, SPARQL provides construct rdf:type to constrain the class of the predicate's subject or object. Let us look at a simple query, Q = Car_Rental $\sqcap$ City.$_{=}$\{Lausanne\}$\sqcap$ hasDeposit.$_{<}$200. In English, it means "find all car_rental services in Lausanne city, which have deposit less than 200CHF".

```
SELECT ?x ?address
WHERE { ?x rdf:Type CO:Car_Rental .
        ?x CO:city "Lausanne" .
        ?x CO:deposit ?deposit .
        Filter (?deposit < 200 ) .
        ?x CO:address ?address .
      }
```

SPARQL supports a set of data-types and operators used to construct constraints. They not only cover data-types, functions/operators specified in XQuery 1.0 and XPath 2.0 but also include some new features, e.g. some extensible value testing (i.e. to test the geographical distance between two points). Besides the basic constraints on values and classes, SPARQL specifies *solution modifiers* which enable to transform a list of solutions in multiple ways. Here is the SPARQL syntax on applications of these solution modifiers:

SelectQuery ::= 'SELECT' ('DISTINCT' | 'REDUCED')? (Var+ | '*')
  DatasetClause* WhereClause SolutionModifier
SolutionModifier ::= OrderClause? LimitOffsetClauses?
LimitOffsetClauses ::= (LimitClause OffsetClause? | OffsetClause LimitClause?)
OrderClause ::= 'ORDER' 'BY' OrderCondition+
OrderCondition ::= (('ASC' | 'DESC') BrackettedExpression) | (Constraint | Var)
LimitClause ::= 'LIMIT' INTEGER
OffsetClause ::= 'OFFSET' INTEGER

DISTINCT and REDUCED The DISTINCT solution sequence modifier D ensures solutions in the sequence are unique, i.e. D(S) = $\{S'_1, \ldots, S'_n\}$, for any $S'_i, S'_j \in$ D(S), $S'_i \neq S'_j$, similar to DISTINCT keyword in SQL. In contrast, REDUCED solution modifier simply permits the duplicates to be eliminated, i.e. the solutions can have one or n duplicate sequences which are no more than the cardinality of the solution set. REDUCED solution modifier is not commonly-used since it can not be used with aggregation functions on the results set.

ORDER BY The ORDER BY clause establishes the order of a solution sequence. It is composed of an expression (i.e. a variable or a function) and an optional order modifier either ASC() or DESC(). Given an order condition O(S, C) = $\{S'_1, \ldots, S'_n\}$, where $S'_i \succeq_c S'_j$ or $S'_i \sim_c S'_j$, $1 \leq i < j \leq n$. The semantics of operator $\succeq_c$ is similar to the Ordered-property (see Definition **??**) defined in chapter 6. The semantics of multiple order conditions (Order By $C_1, C_2, \ldots, C_m$) are regarded as prioritized composition described in [Kie02]:

$S'_i \succ_{C1,C2} S'_j \equiv S'_i \succ_{C1} S'_j \vee (S'_i \sim_{C1} S'_j \wedge S'_i \succ_{C2} S'_j)$

In addition, using Order By clause in a Select form can only order the sequence of results. When it is combined with LIMIT and OFFSET, it will return a different slice of the solution sequence.

LIMIT It puts an upper bound on the number of solutions returned, i.e. LIMIT(S, n) = $\{S'_1, \ldots, S'_n\}$. When n is greater than the limit number of solutions, at most the limit number of solutions will be returned.

OFFSET It causes the solutions generated to start after the specified number of solutions. It functions similarly to the cursor in SQL. When LIMIT, OFFSET and ORDER BY are combined together with a SELECT form, it returns a subset of solution sequences in a certain order.

Query Forms. SPARQL offers four query forms, i.e. SELECT, CONSTRUCT, ASK, and DESCRIBE. The SELECT form returns all or a subset of variables bound in a query pattern match. Different solution modifiers can be combined and applied to modify the results. The CONSTRUCT form returns an RDF graph constructed by substituting variables in a set of triple templates. The result is an RDF graph formed by taking each query solution in the solution sequence, substituting variables in the graph template, and combining triples into a single RDF graph using set UNION. It is particularly useful to rename the properties with the same semantics, for instance, *open time* and *work hours* from two RDF graphs (e.g. from two data profiles). The ASK form returns a Boolean indicating if a query pattern has a solution, e.g. ASK{?x, CO:firstName, 'Shijun'} and its answer is *Yes*. The DESCRIBE form returns a RDF graph that describes the resource found, e.g. return the explicit IRIs of a resource, identify a resource with a property and its variable. SPARQL also provides other functions, which are out of the scope of this introduction and whose description can be found on the W3C website.

The conjunctive query can be easily transformed into SPARQL format. In the example above, we directly transform the begin-time/end-time constraints (i.e. Thursday 19:00, Saturday 19:00) to the xsd:datetime data-type, by referring to the local calendar. The spatial comparison can be done by SPARQL or a GIS database. The final results will be returned in the ascending order of rental price. However, basic SPARQL cannot express the logic semantics of some axioms, as well as existential($\exists$) and universal($\forall$) restrictions on properties. Some efforts have been made on integrating SPARQL querying capabilities and reasoning functionalities on top of OWL-DL, e.g. SPARQL-DL in [SP07]. We employ their abstract syntax in translating

Table 8.1: The basic translation from abstract OWL-DL syntax to SPARQL-DL syntax.

Query Atom p_i in Q	Translation to SPARQL graph form
Type(a, C)	<a, rdf:type, C>
PropertyValue(a, p, v)	<a, p, v>
SameAs(a, b)	<a, owl:sameAs, b>
DifferentFrom(a, b)	<a, owl:differentFrom, b>
SubClassOf(C_1, C_2)	$<C_1$, rdfs:subClassOf, $C_2>$
EquivalentClass(C_1, C_2)	$<C_1$, owl:equivalentClass, $C_2>$
DisjointWith(C_1, C_2)	$<C_1$, owl:disjointWith, $C_2>$
ComplementOf(C_1, C_2)	$<C_1$, owl:complementOf, $C_2>$
SubPropertyOf(p_1, p_2)	$<p_1$, rdfs:subPropertyOf, $p_2>$
EquivalentProperty(p_1, p_2)	$<p_1$, owl:equivalentProperty, $p_2>$
ObjectProperty(p)	<p, rdf:type, owl:objectProperty>
DataProperty(p)	<p, rdf:type, owl:dataProperty>

our conjunctive query to the SPARQL graph as shown in Table 8.1. Most of the commonly used operators in our queries, such as =, <, > and temporal operators can be transformed in SPARQL format. For instance, as shown in Example 8.1, FILTER can express constraints on date-time, numeric and string values. However, we do not discuss issues on translating predicate p with concrete domains in SPARQL. Discussion of problems relevant to DL with concrete domains and proposed solution can be found in the literature, e.g. [BKW03].

Example 8.1. The Original query Q := Car_Rental $\sqcap$ Near.$_=$\{Ouchy\} $\sqcap$ From.$_=$ \{Thursday 19:00\} $\sqcap$ To.$_=$ \{Saturday 19:00\} $\sqcap$ CarType.$_=$\{Automatic\}, and order the results by ascending price. This query is expressed in SPARQL as follows.

```
SELECT ?x ?address ?p
WHERE { ?x rdf:class CO:Car_Rental .
        ?x CO:near "Ouchy" .
        ?x CO:from ?begin .
        ?x CO:end ?end  .
        ?x CO:price ?p  .
        ?x CO:cartype "Automatic" .
        Filter (?begin = "2007-10-18 19:00"^^xsd:datetime) .
        Filter (?end = "2007-10-20 19:00"^^xsd:datetime) .
        ?x CO:address ?address .
      } ORDER BY ?p
```

8.3 Query Relaxation

Query relaxation is not a new topic. In [CC94], Chu et. al. proposed to use a type abstraction hierarchy to relax the variables or concepts in the original query to get cooperative answers. In [GGM92], Gaasterland et. al. proposed the notion of *query relaxation* and applied it for expanding deductive database and logical programming queries. Based on their categorization of types of query relaxations, we extend them with LBS's specific needs and present our extensions in RDF triples as follows:

1. To find the synonym/acronym/functionally similar concept or broaden the variable with certain constraint:

   (?park, along, 'Leman Lake') $\Longrightarrow$ (?park, along, 'Geneva Lake'),

   (?bus, at, 'Ecublens') $\Longrightarrow$ (?metro, at, 'Ecublens'),

   (?price, $\leq$, 200) $\Longrightarrow$ (?price, $\leq$, 300).

2. To broaden the domain of a variable:

   (?car_repair, specializeIn, 'BMW') $\Longrightarrow$ (?car_repair, specializeIn, ?x) $\wedge$ (?x, type, CarMark),

3. To relax the role:

   (?car_repair, specializeIn, ?y) $\Longrightarrow$ (?car_repair, canRepair, ?z)

4. To relax the concept (subject or object):

   (?car_repair, specializeIn, RaceCar) $\Longrightarrow$ (?car_repair, specializeIn, Car).

   (?x, type, Souvenir_shop) $\wedge$ (Souvenir_shop, subclass, Shop) $\Longrightarrow$(?x, type, Shop)

According to the classification of types of the query tuple relaxation, we can find that to *broaden the variables with certain constraint* (in type 1) is a specialization of to *broaden the domain of the variable*(in type 2). Limiting the constraint over query relaxation is universal in human discourse, it can avoid redundant or overabundant answers. A typical example is to book air-tickets online. When a user can not find ticket with given departure/arrival dates, (s)he will be prompted to relax the bounds of arrival/departure with certain constraints (e.g. ± 3 days) rather than to the whole domain. In this section, we will start with the definition of query triple relaxation and show how it can be applied in our work. Further, we discuss the issues concerning restricting and ranking triple relaxation. Then we present the query relaxation profile which specifies the strategy to control the query relaxation, and illustrate our approach with examples.

8.3.1 The Query Triple Relaxation vs. Relaxed Query

In conjunctive query Q = (t_1, …, t_n), for each triple t_i in Q, it can be relaxed to t_i', please notice that there may exist $t_i = t_i'$, i.e. t_i can not be relaxed. Let us give the definition of triple relaxation as follows:

DEFINITION 8.1. Triple Relaxation. *A triple relaxation is a mapping from a RDF-triple t to another RDF-triple t', denoted as $t \preceq_t t'$, where $t = (a, p, b)$, $t' = (a', p', b')$, and $\preceq_t$ is in one or a combination of of the following formats:*

- $a = a'$, $p = p'$, $p \in$ *DataProperty, it holds* $\{x|(x, p, b)\} \sqsubseteq \{x|(x, p, b')\}$;
- $a = a'$, $p \in$ *DataProperty, and* $(p, subPropertyOf, p')$;
- $a = a'$, $p = p'$, $p \in$ *ObjectProperty*, $b \in$ *Class, and* $(b, subClassOf, b')$;
- $b = b'$, $p = p'$, $p \in$ *ObjectProperty*, $b \in$ *Class, and* $(b, subClassOf, b')$;
- $t \simeq t'$.

The query triple relaxation above is specified according to the classification of types of triple relaxation. The first format refers to the variable substitution, e.g. a synonym/hypernym for a string variable, a region with larger range for a spatial landmark, a relaxed number/date for bound numeric/data variable etc. It is very common in cooperative query answering. The second relaxation stands for the property substitution, i.e. substitute the property in the triple with its super-property. An problem may arise in property substitution, i.e. domains of b and b' can be different. The third and fourth formats describe the concept subsumption, i.e. either the subject or the object is replaced with its super-class. Please note that we allow t and t' to be a pair of equivalent (or similar) triples, i.e. "$\preceq$" can be reflexive. We use "$\prec$" to distinguish it from the irreflexive triple relaxation.

DEFINITION 8.2. Relaxed Query. *A relaxed query is a query Q_{RELAX} with 1 to n triple relaxations over Q, where $Q = (t_1, \ldots, t_n)$, $Q_{RELAX} = (t'_1, \ldots, t'_n)$, for all triple relaxations $t_i \preceq t'_i$ ($1 \leq i \leq n$), at least $\exists t_j \prec t'_j$ ($1 \leq j \leq n$).*

More often, there may exist multiple directions for relaxation on a single query and even on a single triple. Consequently, for a query, its relaxed queries can become huge, in particular when the original query has multiple constraints. It is important to restrict, validate and rank the possible relaxations so as to make the size of answers set reasonable. For instance, assume a service class Car_Repair in the core ontology and a triple relaxation as follows:

(?car_repair, specializeIn, RaceCar) $\Longrightarrow$ (?car_repair, specializeIn, Car). (1)

(?car_repair, specializeIn, RaceCar) $\Longrightarrow$ (?car_repair, Repair, RaceCar). (2)

With some domain knowledge, we can tell from the relaxation (2) is more meaningful in the above two relaxed queries. With each triple relaxation, we can obtain from zero to multiple relaxed queries. The potential relaxed queries can be presented to end-users so as enable them to choose the desirable one(s). Alternatively, LBS can validate and rank them, based on certain integrity constraints, heuristics and user constraints in user profiles.

Restrictions on Relaxed Queries. In Algorithm 8, we have presented how to discover the set of hard constraints on a given service from a user's profile, denoted as *CQ(S, uID)*. CS(S, uID) refers to all explicit constraints on a given service S, and these constraints are expressed on properties over S. Let us assume the original query is Q (and the user uID asks the query Q), all relaxed queries over Q is $Q_{RELAX} = \{Q_1, \ldots, Q_m\}$, $\forall Q_i \in Q_{RELAX}$, if Q_i is incompatible with CQ(S, uID), remove Q_i from Q_{RELAX}. For instance, a user specifies "dislike MacDonald's" in her/his profile, if a relaxed query $Q_i = (t'_1$, ..., <?x, hasName, 'MacDonald's'>, ...), Q_i will be removed from the relaxed queries. In addition, at the query formulation phase, we may permit users to explicitly specify what property/value/triple can not be relaxed. This approach can effectively enforce the restrictions on query relaxation from the user's view point. Approaches of applying *query relaxation profiles* to restrict query relaxations will be discussed in next subsection.

Validations on Relaxed Queries. For any relaxed query, there exists at least one irreflexive triple relaxation as defined in Definition 8.2. In other words, the relaxed query is different from the original one, it is necessary to check integrity constraints on the relaxed one. There exist a set of integrity constraints for a given service class. The integrity constraints are categorized into two types: integrity constraints on a single triple, e.g. the data-value range of a dataProperty; integrity constraints between triples, e.g. to book an air-ticket, the departure time must be later than the arrival time, for a flight from Geneva to Beijing, the departure date and arrival date can not be same. Integrity constraints ensure that relaxed queries are valid with the specification of service class and are consistent with the original query.

Heuristics on Query Relaxation. In cooperative answering systems [CC94][Gaa97], to correct misconception was deemed as the most important reason for relaxation over a query. Authors also pointed out that the misconception often happens due to the user's incomplete knowledge on the database schema. It may result in the sure failure of a query, redundant search space, and a huge number of answer substitutions. They also presented the basic strategies on query relaxation for a misconception:

1. Relaxation of misconception receives priority over other relaxations.
2. When multiple misconception are available for relaxation, diverse precedence levels applies on them.
3. When multiple misconception have same relaxation priority, the misconception can be chosen according to the depth of relaxation's derivation.

8.3.2 The Definition of Query Relaxation Profile

The query relaxation profile (RP) is regarded as a working repository to handle query relaxation whenever query relaxation is needed. Each *RP* corresponds to a specific service class in the core ontology. It has two-folded functionalities. On the one hand, a *RP* contains a set of relaxation rules, i.e. what property of the service class can be relaxed and how to control relaxations. On the other hand, each RP is uniquely associated with a query profile (QP), and both of them have the common service class S. A query profile refers to a set of queries on the service class S which have been processed by LBS, i.e. who and in which context asks what service, possibly with services chosen by the user. Simply, QP is uniquely identified with the service class S and denoted as a data tuple such as QP = (S, Q_S) where $Q_S = \{<q, c_q, uID, A>\}$, q is the original query delivered by the user uID, c_q refers to the context relevant to q and uID, and A is the answer (or answer set) chosen by the user uID. A query profile can be extracted from the query log files and organized by the service class S. Query profiles can assist LBS to know what questions were asked by the given user within a certain time-frame, and to further understand the real needs of the user and make suitable recommendation and ranking. In return, the analysis of query profiles can help to modify the query relaxation profiles to adapt the relaxation rules to the real needs of users. Moreover, query profiles can help to find out the collaborative answers for a certain user with data mining techniques. However, it is not our main concern to cluster query records of different users for collaborative query answering, but relevant efforts can be found in [GGM92] [SB06].

Now let us have a closer look at the query relaxation profile. In earlier this chapter, we have presented there may exist a large number of relaxed queries for a query Q. Hence, the relaxation profile mainly provides a set of heuristic rules for query relaxation on a given service class S. For instance, the triple relaxation (?car_repair, specializeIn, RaceCar) $\Longrightarrow$ (?car_repair, specializeIn, Car) has little/no sense to constrain the answer, since all car_repair services specialize in repairing cars. In addition, a ranking algorithm is needed to evaluate the potential relaxed queries Q_{relax}.

DEFINITION 8.3. Query Relaxation Profile. *A query relaxation profile RP is a data tuple for a given service class S in CO, such that RP = (S, R(t), F), where R(t) is a set of relaxation rules on triple t, $\forall t$ over S; F is a ranking function to order the priority over R(t).*

Example 8.2. The original query Q = (?x, type, Car_Rental)$\wedge$ (?x, PickupTime, ?t1)$\wedge$(?t1, at, '2007-10-12 19:00')$\wedge$(?x, ReturnTime, ?t2)$\wedge$(?t, at, '2007-10-16 21:00')$\wedge$ (?x, PickupPlace, ?l)$\wedge$(?i, near, "Ouchy")$\wedge$(?x, hasCarType, 'Automatic')$\wedge$(?x, hasCarMark, BMW).

RP := (Car_Rental, R(t), F)
R(t):= {(?x, PickupTime, ?t) $\Longrightarrow$ (?x, PickupTime, ?t$\pm t_y$)$\wedge$(t_y, $\leq$, 2hours), (1)
(?x, ReturnTime, ?t) $\Longrightarrow$ (?x, ReturnTime, ?t$\pm t_z$)$\wedge$(t_z, $\leq$, 2hours), (2)
(?x, PickupPlace, ?l) $\Longrightarrow$ (?x, PickupPlace, District), (3)
(?x, PickupPlace, ?l) $\Longrightarrow$ (?x, PickupPlace, ?l')$\wedge$(?l', covers, ?l), (4)
(?x, hasCarMark, BMW) $\Longrightarrow$ (?x, hasCarMark, CarMark), (5)
(?x, type, Car_Rental) $\Longrightarrow$ (?x, type, Minibus_Rental). (6) }.
F:= ((relax(4)$\succeq$relax(3))$\succeq$relax(5)$\succeq$relax(2)$\simeq$ relax(1)$\succeq$relax(6))

In the above example, we show the definition of a relaxed profile, we assume the query Q make constraints on all properties of the service class Car_Rental. In the example, we observe only property hasCarType has not been specified within relaxation rules, because it is regarded as a non-relaxable property. In the ranking function, we rank triple relaxations according to the respective relaxed triple t_i, where $\succeq$ means have precedence to make relaxation over other triple relaxations, $\simeq$ means have equivalent relaxation priority. Further, the ranking function can be more complex, e.g. metrics-based function. But it is out of the scope of this thesis.

Discussion. When the user profile is incomplete, the query history and collaborative user profiles will play an important role in query reformulation and relaxation. On the basis of mining results of user queries, for a given user, some preferences (or relaxation constraints) on a certain service can be discovered, for instance, a user often chooses menu in certain price range and certain cuisine style. The inferred user information is also stored in user profile as its complement. In addition, for a given user, the more recent queries are regarded as type of context information to help LBS to deliver appropriate services to the user. For instance, if the user visited a book shop one hour ago, the information from the same shop will not be delivered again. The collaborative user profiles refer to user profiles of those users who have common interests or share common characteristics with the current user. Consequently, LBS assume their user profiles can be taken into account as the useful complement of current user's profile in particular for a new LBS user.

8.4 Chapter Summary

Query processing in LBS opens a plethora of new research directions in terms of query handling and optimization. Different from the well-structured data in database systems, the data in LBS origins from heterogeneous sources so that they are naturally presented in the format of RDF-graphs. Consequently, emerging RDF-based query languages have been able to provide certain data querying functionalities, such as sort, group results etc. However, it still calls upon new strategies, e.g. consistency checking and spatio-temporal support. In this chapter, we concentrated on issues of query answering and relaxation in LBS. With the SPARQL, we described a vision of query answering in LBS, and explained how to transform the conjunctive predicate to SPARQL syntax. Then we discussed the critical issues of query relaxation and proposed to use *query relaxation profiles* to specify the relaxation rules for each service class. In addition, by analyzing the query data, query relaxation profiles may evolve to apt to the real needs of LBS users, similar to the core ontology's evolution for LBS. However, we did not deploy the query processing algorithms, we anticipate that LBS applications will benefit from our strategy of query processing and open new research issues in LBS.

Chapter 9

Conclusion

9.1 Contributions of the Thesis

Location-based Services (LBS) can be expected to become extremely popular in the very near future. Many factors converge in substantiating this belief. On the hardware side, the emergence of sophisticated mobile devices enables users on the move to easily dialog with an information service. Moreover, the development of sensors makes data readily available about the current status of the world in user's surroundings. On the software side, the promises of the semantic web, in terms of handling and exchanging heterogeneous information formats and contents, enable LBS to provide mobile users with easy access to an over-abundant amount of information they may need to organize their out-of-office activities. Given this trend and potential, it makes sense to investigate how the capabilities of current LBS can be improved to turn them into knowledgeable information providers, able to assist users on the move with better services such as contextualized and personalized local information. The mobility framework is inherently characterized by high dynamicity: focus on information evolves according to user movement, and local context has to be continuously updated to always represent the current status of the world. This makes current static and ad hoc solutions poorly suited for a long term effort and temporal scalability (i.e. long lasting services). Instead, fully flexible information frameworks have to be designed to be able to adjust LBS to changing mobile users with changing requirements in a changing world.

The essence and objective of this thesis is to explore the semantic issues in designing an LBS data infrastructure tailored for supporting contextualized and personalized services. Such an analysis of semantic aspects in LBS, following a generic, application independent approach, remained to be done. The outcome of our analysis, and the major contribution of this work, is a proposal for a specific organization of the knowledge an LBS has to handle. The proposal includes a methodology to build the different components of the LBS knowledge infrastructure, a characterization of the salient features of each component, and a specific approach to organize the interrelationships among the components so that the contextualization and personalization goals can be effectively achieved.

In this document, we firstly review the evolution of the LBS concept from positioning services to recent mobile information services. We then present our overall vision of future LBS, identifying the target functionalities that we believe are important and can be achieved by focusing on issues such as the ones we address in this thesis, i.e. the design of a semantic framework, knowledge representation, personalization, context-awareness, a simple and efficient query reformulation approach, and query answering and relaxation strategies. Finally, we introduce our LBS architecture and make a systematic and comprehensive survey on related work of each component in the framework.

In Chapter 3 we propose a modular core ontology as the main body of knowledge representation, and the modular profiles as distributed data repositories interacting with the LBS core. Within the core ontology, besides adopting the basic definitions from databases and ontologies, we focus on proposing a set of new constructs that enhance the semantics of classes, properties and relations specifically involved in supporting LBS. According to their functionality and characteristics, all dedicated concepts are classified as belonging to the *Service*, *Context*, and *User* modules. Thanks to LBS-driven mappings between the concepts in the core ontology and the ones in each data source, we allow heterogeneous data sources to maintain and evolve their content in an autonomous and consistent manner. Two additional modules, the spatial and the temporal modules, provide basic but essential support of data types and variables (i.e. landmarks) to express the spatio-temporal features and constraints for the concepts in the core ontology. Lastly we briefly investigate the evolution issues for the core and mapping ontologies, illustrating the common update operations and their potential effects on the LBS ontologies.

We aim at handling the three main components of an LBS, the service, context, and user data, in a similar fashion. The concept's semantic definition and metadata are centrally maintained in the core ontology, while all data instances are stored separately in their respective profiles. However, we also address specific features of each module from the data's semantics viewpoint. In the service module, for example, we define a set of relationships to facilitate query processing. For instance, *functional similarity* relationships assist in discovering alternative services when perfect service matching cannot be achieved. In addition, we detail the service mapping process in a top-down manner, and illustrate the definition of *mapping* with examples.

Context is universal and many concepts in an LBS are context-sensitive. Our approach suggests a generic classification of context concepts into five categories representing different knowledge domains. Although we advocate specific arguments in support of this classification, the classification itself is proposed as a possible but not mandatory general view of context data in LBS. What is important is that the classification is built and maintained using a sound methodology that guarantees that the chosen context data is indeed both useful and needed.

In the user module, we use the composition relationship to separate user's complete profile from her/his potentially many contextual profiles, and specify ways to represent the specificities of user properties in user profiles.

Finally, to express what context data is meant for, and to support contextualizing information on services as well as on users, two inter-module links are introduced that explicit the *determine* and *influence* semantics that interrelates concepts from different modules. To complete this part of our proposal, inter-concepts relationships between modules are complemented with a *property relevancy association* that improves the LBS knowledge about relevance of concepts between different modules.

Query reformulation and processing are challenging topics for any information system. In the mobile environment, due to device and network limitations, query formulation becomes more cumbersome than in the fixed computing environment. We therefore propose a simple and intuitive way for users to formulate a query, i.e. as <what, where, when, what-else> tuples, and show how to better understand the terms in a query thanks to terminological support, before transforming the query tuples into conjunctive queries. Using the SPARQL query language, we show that the conjunctive queries can be smoothly translated to SPARQL-compatible format. Last, we discuss the query relaxation issues using a rule-based approach and show its feasibility with some scenarios.

9.2 Future Work

Our proposal for an LBS semantic data infrastructure and its analysis in terms of set up methodology and design considerations should enable a systematic implementation-oriented development of the various components in support of knowledgeable LBS. This is obviously the next task on the agenda towards the new generation of LBS, the one that would really provide users the information services they look for. Detailed technical specifications remain to be elaborated once the underlying technical infrastructure would be defined: which representation formalism, which query language, which data management system, which data exchange protocols would best suit the implementation of the objectives. Most likely, choices will here be different from one team to another. In any case, the technical specifications task, not to mention the implementation task, encompasses so many diverse aspects that it is indeed the work for a team and not for a single researcher.

Beyond what we aimed to cover with our analysis of semantic requirements, a wide variety of challenges remain to be addressed for Location-based services to emerge as powerful information services. Examples include standardization of annotations for web-based information, acknowledged protocols for privacy protection, and intelligent aggregation of sensor network data, just to name a few.

Implementation-Oriented Issues. The very first effort is to set up a trial prototype to examine the algorithms designed in our work. The prototype shall have the capability to handle a large population of service data, context data and user data. It brings up a big challenge to collect the relevant data is a challenge in itself as it has to be a dynamic process in a dynamic computing environment. A variety of knowledge extraction techniques are needed to get data from many sources that are heterogeneous both in format and in content. While many of these techniques are available or being investigated, very rare results have been achieved in terms of extracting spatial and temporal data. This is an area where much more research is needed. Validation of the prototype calls for the realization of many case studies to evaluate users' experiences and satisfaction on using such a system. In particular, in query processing and relaxation we only introduced a simple way to compute and rank the metrics of the result set. We believe more complex metrics-based algorithms can be employed in ranking results and in computing concepts similarity.

Logical Formalism Issues. In our work, we defined our core ontologies and mapping ontologies based on OWL-DL. However, in the world of real systems, more *closed world* assumptions are advocated to control the scalability of ontological reasoning and searching in order to respond user queries in a rapid and effective manner. The OWL community is actively making efforts to enhance the pragmatics of OWL in semantic web applications and to explore a better query language to enable to quickly find out the data among a large scale of ontology repositories. However, functionality needed for management of spatial and spatio-temporal data represent a serious challenge that may simply inhibit the design of fully OWL-based LBS. Therefore, hybrid solutions are likely to be needed to combine the reasoning benefits of logical languages with the effective services provided by DBMS and GIS. This is another promising area for future research.

Trust and Privacy Issues. Security and trust are very fundamental and even radical issues in accomplishing LBS from both the user and service providers' viewpoints. Many endeavors have been made in fostering the security and privacy protection in location-based services setting. While we have tentatively explored the privacy issues in chapter 6 in terms of privacy properties and user's rules definition, further work is needed on integrating trust and security management within the whole infrastructure so as to encourage the popularity and proliferation of LBS in reality.

Bibliography

[AAH$^+$97] G.D. Abowd, C.G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks*, 3:421–433, 1997. 21, 25

[AJPS07] L. Al-Jadir, C. Parent, and S. Spaccapietra. Ontomind: Reasoning with large owl ontologies stored in relational databases. Technical Report LBD-REPORT-2007-005, EPFL, 2007. 101, 170

[All83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. 116, 161

[AR02] J. Allan and H. Raghavan. Using part-of-speech patterns to reduce query ambiguity. In *proceedings of the 25th Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR '02)*, 2002. 63, 153, 155

[AW00] R. Agrawal and E.L. Wimmers. A framework for expressing and combining preferences. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD 00)*, pages 297–306, 2000. 121

[BB05] M. Bazire and P. Brézillon. Understanding context before using it. In *proceedings of the Internatioanl and Interdisciplinary Conference on CONTEXT*, pages 29–40, 2005. 96

[Ber96] P.A. Bernstein. Middleware: a model for distributed system services. *Communications of the ACM*, 39(2):86–98, 1996. 17

[BGMP00] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Focused web searching with pdas. In *proceedings of the 9th International World Wide Web Conference*, pages 213–230, 2000. 24

[BGvH$^+$03] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-owl: Contextualizing ontologies. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, pages 164–179, 2003. 135

[BJ05] S. Bessler and O. Jorns. A privacy enhanced service architecture for mobile users. In *proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'05)*, pages 125–129, 2005. 17

[BKW03] F. Baader, R. Küsters, and F. Wolter. *Extensions to description logics*, pages 219–261. Cambridge University Press, New York, NY, USA, 2003. 178

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American Magazine*, May, 2001. 24

[BLR03] A. Borgida, M. Lenzerini, and R. Rosati. *Description logics for databases*, pages 462–484. Cambridge University Press, New York, NY, USA, 2003. 169

[BPvS$^+$04] T. Broens, S. Pokraev, M. van Sinderen, J. Koolwaaij, and P. Dockhorn-Costa. Context-aware, ontology based, service discovery. In *proceedings of the European Symposium on Ambient Intelligence 2004 (EUSAI'04)*, 2004. 25

[BR00] P.A. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *proceedings of the 19th International Conference on Conceptual Modeling (ER'00)*, pages 1–15, 2000. 92

[Bro02] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002. 147

[CC94] W. W. Chu and Q. Chen. A structured approach for cooperative query answering. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):738–749, 1994. 178, 181

[CCC$^+$04] A. Calí, D. Calvanese, S. Colucci, D. Di Noia, and F.M. Donini. A description logic based approach for matching user profiles. In *proceedings of the Description Logic Workshop (DL 2004)*, 2004. 28, 121

[CDM$^+$00] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'00)*, pages 17–24, 2000. 5, 148

[CDM$^+$03] A. Cole, S. Duri, J. Munson, J. Murdock, and D. Wood. Adaptive service binding middleware to support mobility. In *proceedings of the 23rd international Conference on Distributed Computing Systems(ICDCSW'03)*, 2003. 17

[CF03] H. Chen and T. Finin. An ontology for context-aware pervasive computing environments. *Knowledge Engineering Review,Special Issue on Ontologies for Distributed Systems*, 2003. 97

[CG00] J. Chaffee and S. Gauch. Personal ontologies for web navigation. In *proceedings of the 9th international Conference on information and Knowledge Management*, pages 227–234, 2000. 26

[CGL01] D. Calvanese, G. Giacomo, and M. Lenzerini. Ontology of integration and integration of ontologies. In *proceedings of Description Logic Workshop (DL 2001)*, pages 10–19, 2001. 87, 89

[Cho03] Jan Chomicki. Preference formulas in relational queries. *ACM Transactions Database Systems*, 28(4):427–466, 2003. 120

[CK02] G. Chen and D. Kotz. Solar: An open platform for context-aware mobile applications. In *proceedings of the First International Conference on Pervasive Computing (Pervasive 2002)*, pages 41–47, 2002. 99

[CM03] M. Crubèzy and M.A. Musen. *Handbook on Ontologies*, chapter Ontologies in support of problem solving., page 321342. Springer Publisher, 2003. 87

[CP07] J.-P. Calbimonte and F. Porto. Extending owl with explicit dependency. Technical Report LBD-REPORT-2007-002, EPFL, 2007. 53

[CSM$^+$01] K. Cheverst, G. Smith, K. Mitchell, A. Friday, and N. Davies. The role of shared context in supporting cooperation between city visitors. *Computers & Graphics*, 25:555–562, 2001. 26

[DA99] A.K. Dey and G. Abowd. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, 1999. 5, 21

[DB03] T. D´Roza and G. Bilchev. An overview of location-based services. *BT Technology Journal*, 21(1), 2003. 2

[DCMF99] N. Davies, K. Cheverst, K. Mitchell, and A. Friday. Caches in the air: Disseminating information in the guide system. In *proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 11–19, 1999. 4, 16, 25

[Dey01] A.K. Dey. Understanding and using context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7, 2001. 21, 22, 96, 97, 99

[dFFRV05] C. di Flora, M. Ficco, S. Russo, and V. Vecchio. Indoor and outdoor location based services for portable wireless devices. In *proceedings of First international Workshop on Services and infrastructure For the Ubiquitous and Mobile internet (ICDCSW'05)*, 2005. 17

[DG03] T. Candebatand C.-R. Dunne and D. Gray. The orient platform: A secure infrastructure for location based services on the internet. In *proceedings of the IEI/IEEE Irish Telecommunications System Research Symposium*, 2003. 17

[dIL01] D. Lopez de Ipina and S-L. Lo. Locale: a location-aware lifecycle environment for ubiquitous computing. In *proceedings of the 15th International Conference on Information Networking*, pages 419–426, 2001. 17

[DMQ03] D. Dou, D. McDermott, and P. Qi. Ontology translation on the semantic web. In *proceedings of the international Conference on Ontologies, Databases and Applications of Semantics*, 2003. 87

[Do06] H.H. Do. *Schema Matching and Mapping-based Data Integration.* PhD thesis, University of Leipzig, Germany, 2006. 89

[Dou04] P. Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, 2004. 21

[DSA01] A.K. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2-4):97–166, 2001. 22, 99

[ES05] M. Ehrig and Y. Sure. Foam - framework for ontology alignment and mapping - results of the ontology alignment evaluation initiative. In *Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies*, 2005. 87

[FD92] P.W. Foltz and S.T. Dumais. Personalized information delivery: an analysis of information filtering methods. *Communications of the ACM*, 35:51–60, 1992. 26

[FJA05] F. Fu, C.B. Jones, and A.I. Abdelmoty. Ontology-based spatial query expansion in information retrieval. In *proceedings of the international Conference ODBASE*, 2005. 30

[FM02] A. Franz and B. Milch. Searching the web by voice. In *proceedings of the 19th international Conference on Computational Linguistics - Volume 2 (Taipei, Taiwan, August 24 - September 01, 2002). International Conference On Computational Linguistics.*, 2002. 148

[Gaa97] T. Gaasterland. Cooperative answering through controlled query relaxation. *IEEE Expert: Intelligent Systems and Their Applications*, 12(5):48–59, 1997. 181

[Gar04] G. Gartner. Location-based mobile pedestrian navigation services - the role of multimedia cartography. In *proceedings of International Joint Workshop on Ubiquitous, Pervasive and Internet Mapping (UPIMap2004)*, 2004. 2, 16

[GBE$^+$00] R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transaction on Database Systems*, 25(1):1–42, 2000. 7

[GF05] A. Gershman and A. Fano. Examples of commercial applications of ubiquitous computing. *Communications of the ACM*, 48(3):71, 2005. 1

[GGM92] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3), 1992. 178, 181

[GJM01] K. Govindarajan, B. Jayaraman, and S. Mantha. Preference queries in deductive databases. *New Generation Computing*, 19(1):57–86, 2001. 121

[GPFLCG04] A. Gòmez-Pèrez, M. Fernández-Lòpez, and Ò. Corcho-Garcia, editors. *Ontological Engineering*. Springer Publisher, 2004. 23

[GPH05] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005. 170

[GPS06] B.C. Grau, B. Parsia, and E. Sirin. Combining owl ontologies using epsilon-connections. *Journal of Web Semantics*, 4(1):40–59, 2006. 132, 134

[Gru93] T.R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, 5:199–220, 1993. 19, 23

[GS01] P.D. Gray and D. Salber. Modelling and using sensed context information in the design of interactive applications. In *proceedings of the 8th IFIP Conference on Engineering for Human-Computer Interaction*, pages 317–335, 2001. 22, 65

[GT04] J. Gong and P. Tarasewich. Guidelines for handheld mobile device interface design. In *proceedings of the Device Interface Design Annual Meeting (DSI 2004)*, 2004. 30

[Gua97] N. Guarino. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. In *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, pages 139–170. Springer-Verlag, 1997. 23

[HBEV04] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of rdf query languages. In *Proceedings of the Third International Semantic Web Conference*, Hiroshima, Japan, 2004. 170

[Hea99] M. Hearst. *Modern Information Retrieval*, chapter User Interfaces and Visualization. Addison-Wesley Longman Publishing Company, 1999. 144

[HGM04] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wireless Networks*, 10(6):643–652, 2004. 4

[HI04] K. Henricksen and J. Indulska. Modelling and using imperfect context information. In *proceedings of 1st Workshop on Context Modeling and Reasoning (CoMoRea), PerCom'04 Workshop*, pages 33–37, 2004. 23, 65, 97, 98, 99

[HIR02] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In *proceedings of 1st International Conference on Pervasive Computing (Pervasive)*, pages 167–180, 2002. 19, 21, 22, 100

[HK04] S. Holland and W. Kießling. Situated preferences and preference repositories for personalized database applications. In *proceedings of the the 23rd International Conference on Conceptual Modeling (ER'04)*, 2004. 28, 120

[HL04] J-L. Hong and J.-A. Landay. An architecture of privacy-sensitive ubiquitous computing. In *proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189, 2004. 17

[HLA+04] L. Harvel, L. Liu, G.D. Abowd, Y.X. Lim, C. Scheibe, and C. Chatham. Context cube: Flexible and effective manipulation of sensed context data. In *proceedings of 2nd international Conference on Pervasive Computing (PERVASIVE)*, pages 51–68, 2004. 99

[HP04] V. Hristidis and Y. Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *The VLDB Journal*, 13(1):49–70, 2004. 121

[HSK04] M. Hazas, J. Scott, and J. Krumm. Location-aware computing comes of ages. *IEEE Computer*, 37(2), 2004. 7, 16

[HV03] A. Hinze and A. Voisard. Location- and time-based information delivery in tourism. In *proceedings of International Symposium on Advances in Spatial and Temporal Databases (SSTD'03)*, pages 489–507, 2003. 7, 17, 27, 28, 148

[Jac04] H-A. Jacobsen. *Location-Based Services*, chapter Middleware for Location-Based Services. Morgan Kaufmann, 2004. 17

[Jen04] C. Jensen. *Location-Based Services*, chapter Database aspects of Location-based Services, pages 115–145. Morgan Kaufmann, 2004. 17

[JFCP$^+$01] C. Jensen, A. Friis-Christensen, T. Pedersen, D. Pfoser, S. Saltenis, and N. Tryfona. Location-based services - a database perspective. In *proceedings of the Eighth Scandinavian Research Conference on Geographical Information Science*, pages 59–68, 2001. 7, 17, 25

[JKPT04] C.S. Jensen, A. Kligys, T.B. Pedersen, and I. Timko. Multidimensional data modeling for location-based services. *The VLDB Journal (The International Journal on Very Large Data Bases)*, 13(1):1–21, 2004. 25, 99

[JM02] M. Jarrar and R. Meersman. Formal ontology engineering in the dogma approach. In *Proceedings of the Confederated International Conferences CoopIS/DOA/ODBASE*, page 1238 1254, 2002. 41

[JMMN$^+$99] M. Jones, G. Marsden, N. Mohd-Nasir, K. Boone, and G. Buchanan. Improving web interaction on small displays. In *proceedings of the Eighth international Conference on World Wide Web*, pages 1129–1137, 1999. 149

[JMRD03] R. José, A. Moreira, H. Rodrigues, and N. Davies. The around architecture for dynamic location-based services. *Mobile Networks and Applications*, 8(4):377–387, 2003. 17, 25

[JS99] C.S. Jensen and R.T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):36–44, 1999. 161

[JWS81] A. Joshi, B. Webber, and I. Sag, editors. *Elements of Discourse Understanding.* Cambridge University Press, 1981. 145

[Kaa03] E. Kaasinen. User needs for location-aware mobile services. *Personal Ubiquitous Computing*, 7(1):70–79, 2003. 26, 27

[KBM+02] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002. 25

[KHFH01] W. Kießling, B. Hafenrichter, S. Fischer, and S. Holland. Preference xpath: A query language for e-commerce. In *proceedings of the 5th Internationale Konferenz für Wirtschaftsinformatik*, pages 427–440, 2001. 28

[Kie02] W. Kießling. Foundations of preferences in database systems. In *proceedings of the International Conference on Very Large Databases (VLDB'02)*, pages 311–322, 2002. 28, 120, 151, 177

[KK02] W. Kießling and G. Köstler. Preference sql - design, implementation, experiences. In *proceedings of the International Conference on Very Large Databases (VLDB'02)*, pages 990–1001, 2002. 28

[KMK+03] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E-J. Malm. Managing context information in mobile devices. *IEEE Pervasive Computing*, 2(3):42–51, 2003. 99

[KS03] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *Knowledge Engineering Review*, 18(1):1–31, 2003. 85, 89

[KS05] J. Krumm and S. Shafer. Data store issues for location-based services. *IEEE Data Engineering Bulletin*, 28(1):36–43, 2005. 18

[Law00] S. Lawrence. Context in web search. *IEEE Data Engineering Bulletin*, 23(3):25–32, 2000. 96, 146

[LB03] Y.E. Lee and I. Benbasat. Interface design for mobile commerce. *Communications of the ACM*, 46(12):48–52, 2003. 147

[Lei05] H. Lei. Context awareness: a practitioner's perspective. In *proceedings of International Workshop on Ubiquitous Data Management*, pages 43–52, 2005. 22

[Liu99] M. Liu. Deductive database languages: Problems and solutions. *ACM Computing Surveys*, 31(1):27–62, 1999. 147

[LSC03] X. Liu, S. Shekhar, and S. Chawla. Object-based directional query processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):295–304, 2003. 157

[LSI+02] H. Lei, D.M. Sow, J.S. Davis II, G. Banavar, and M. Ebling. The design and applications of a context service. *Mobile Computing and Communications Review*, 6(4):45–55, 2002. 22, 99, 103

[LYM04] F. Liu, C. Yu, and W. Meng. Personalized web search for improving retrieval effectiveness. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):28–40, 2004. 26

[MB98] J. McCarthy and S. Buvac. Formalizing context (expanded notes). *Computing Natural Language, in: CSLI, Lecture Notes*, 81:13–50, 1998. 21

[Mil95] G.A. Miller. Wordnet: a lexical database for english. *Communications of ACM*, 38(11):39–41, 1995. 24

[MMSV02] A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra - a mapping framework for distributed ontologies. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 235–250. Springer-Verlag, 2002. xi, 87, 88

[MPR00] U. Manber, A. Patel, and J. Robison. Experience with personalization of yahoo! *IEEE Transactions on Knowledge and Data Engineering*, 43(8):35–39, 2000. 26

[NK04] N.F. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6(4):428–440, 2004. 65, 66, 87

[Noy04] N.F. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record, Special Issue on Semantic Integration*, 33(4), 2004. 87, 89

[NSD$^+$05] T. Di Noia, E. Di Sciascio, F. M. Donini, A. Ragone, and S. Colucci. Automated semantic web services orchestration via concept covering. In *proceedings of Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1160–1161, 2005. 80

[NUR03] N. Nanas, V. Uren, and A. De Roeck. Building and applying a concept hierarchy representation of a user profile. In *proceedings of the 26th Annual international ACM SIGIR Conference on Research and Development in informaion Retrieval*, pages 198–204, 2003. 26

[OSJ99] R. Oppermann, M. Specht, and I. Jaceniak. Hippie: A nomadic information system. In *proceedings of the 1st international Symposium on Handheld and Ubiquitous Computing*, pages 330–333, 1999. 7

[OTV05] M. Öztürké, A. Tsoukiàs, and P. Vincke. *Multiple Criteria Decision Analysis: State of the Art Surveys*, chapter Preference Modelling., pages 27–72. Springer Verlag, 2005. 27

[PB97] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997. 26

[PPPS03] D. Pierrakos, G. Paliouras, C. Papatheodorou, and C.D. Spyropoulos. Web usage mining as a tool for personalization: A survey. *User Modeling and User-Adapted Interaction*, 13:311–372, 2003. 26

[PPT02] D. Pfoser, E. Pitoura, and N. Tryfona. Metadata modeling in a global computing environment. In *proceedings of the 10th ACM international symposium on Advances in Geographic Information Systems*, pages 68–73, 2002. 25

[PSS08] C. Parent, S. Spaccapietra, and H. Stuckenschmidt. *Modular Ontologies (Eds.).* Springer Publisher, 2008. 35

[PSZ06] C. Parent, S. Spaccapietra, and E. Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications The MADS Approach.* Springer Publisher, 2006. 41, 49, 54, 55, 92, 104, 117, 128, 161

[RB01] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001. 85, 86, 87

[RC03] A. Ranganathan and R.H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *proceedings of the ACM/IFIP/USENIX International Middleware Conference*, pages 143–161, 2003. 97

[RCC92] D.A. Randell, Z. Cui, and A.G. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning*, 1992. 117

[RD98] S. Ramakrishnan and V. Dayal. The pointcast network (abstract). In *proceedings of the 1998 ACM SIGMOD international Conference on Management of Data*, 1998. 26

[RG01] Schwabe D. Rossi, G. and M. Guimaraes. Designing personalized web applications. In *proceedings of the 10th World Wide Web Conference*, 2001. 26

[RK03] R.T. Rust and P.K. Kannan. E-service: a new paradigm for business in the electronic environment. *Communications of the ACM*, 46(6):37–42, 2003. 5

[RL04] D.E. Rose and D. Levinson. Understanding user goals in web search. In *proceedings of the 13th international Conference on World Wide Web (WWW '04)*, pages 13–19, 2004. 146, 147

[RM03] B. Rao and L. Minakakis. Evolution of mobile location-based services. *Communications of the ACM*, 46(12):61–65, 2003. 16

[RSP05] Y. Roussos, Y. Stavrakas, and V. Pavlaki. Towards a context-aware relational model. In *proceedings of the workshop on Contextual Representation and Reasoning*, 2005. 99

[SAJY05] S. Spaccapietra, L. Al-Jadir, and S. Yu. Somebody, sometime, somewhere, something. In *proceedings of international Workshop on Ubiquitous Data Management (UDM'05)*, pages 6–16, 2005. 16, 26

[SAW94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994. 5, 21, 97

[SB06] B. Smyth and E. Balfe. Anonymous personalization in collaborative web search. *Information Retrieval*, 9(2):165–190, 2006. 181

[SBG99] A. Schmidt, M. Beigl, and H. Gellersen. There is more to context than location. *Computer & Graphics*, 23(6):893–901, 1999. 7

[SBLPZ03] B. Schmidt-Belz, H. Laamanen, S. Poslad, and A. Zipf. Location-based mobile tourist services - first user experiences. In *proceedings of the International Conference for Information and Communication Technologies in Travel and Tourism (ENTER)*, 2003. 17

[SBNPZ02] B. Schmidt-Belz, A. Nick, S. Poslad, and A. Zipf. Personalized and location-based mobile tourism services. In *proceedings of the Fourth International Symposium on Human Computer Interaction with Mobile Devices (MobileHCI '02)*, 2002. 7, 25, 27

[SC00] B. Smyth and P. Cotter. A personalized television listings services. *Communications of the ACM*, 43(8):107–111, 2000. 26

[SC03] S. Shekhar and S. Chawla. *Spatial Databases: A Tour.* Prentice Hall Publisher, 2003. 157

[SKB$^+$98] S. Shafer, J. Krumm, B. Brumitt, B. Meyers, M. Czerwinski, and D. Robbins. The new easyliving project at microsoft research. In *proceedings of the Joint DARPA/NIST Smart Spaces Workshop*, 1998. 16

[SLP04] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management*, 2004. 97

[Sot06] A. Sotnykova. *Semantic Validation in Spatio-Temporal Schema Integration.* PhD thesis, EPFL, Swtizerland, 2006. 56, 58, 116

[SP05] B. Shneiderman and C. Plaisant. *Designing the user interface - Strategies for effective human-computer interaction (4th edition).* Addison-Wesley, 2005. 30, 143, 144, 145

[SP07] E. Sirin and B. Parsia. Sparql-dl: Sparql query for owl-dl. In *proceedings of the Third International Workshop on OWL: Experiences and Directions (OWLED 2007)*, 2007. 170, 177

[SPD$^+$08] S. Spaccapietra, C. Parent, M.L. Damiani, J.A. de Macedo, F. Porto, and C. Vangenot. A conceptual view on trajectories. *to appear in Data & Knowledge Engineering*, 2008. 112

[Spi04] S. Spiekermann. *Location-Based Services*, chapter General Aspects of Location-Based Services, pages 9–26. Morgan Kaufmann, 2004. 6

[SS04a] A. Savidis and C. Stephanidis. Unified user interface development: the software engineering of universally accessible interactions. *Journal of Universal Access in the Information Society*, 3:165–193, 2004. 150

[SS04b] S. Staab and R. Studer, editors. *Handbook on ontologies.* Springer Publisher, 2004. 23

[STW93] B.N. Schilit, M.M. Theimer, and B.B. Welch. Customizing mobile application. In *proceedings of the USENIX Symposium on Mobile and Location-independent Computing*, pages 129–138, 1993. 22, 28

[SV04] J. Schiller and A. Voisard, editors. *Location-Based Services*. Morgan Kaufmann, 2004. 7

[SW03] S. Sarker and J.D. Wells. Understanding mobile handheld device use and adoption. *Communications of the ACM*, 46(12):35–40, 2003. 3

[TCG$^+$93] A.U. Tansel, J. Clifford, S.K. Gadia, S. Jajodia, A. Segev, and R.T. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993. 161

[TFPL04] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data (SIGMOD'04)*, pages 611–622, 2004. 7

[Tis01] J. Tisal. *The GSM network: the GPRS evolution, one step towards UMTS*. Willey Publisher, 2001. 1

[TLKT01] T. Tezuka, R. Lee, Y. Kambayashi, and H. Takakura. Web-based inference rules for processing conceptual geographical relationships. In *Proceedings of the 2th International Conference on Web Information Systems Engineering WISE*, 2001. 39, 106

[TMK$^+$06] T. Terada, M. Miyamae, Y. Kishino, K. Tanaka, S. Nishio, T. Nakagawa, and Y. Yamaguchi. Design of a car navigation system that predicts user destination. In *Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)*, pages 145–160, 2006. 7

[TP05] N. Tryfona and D. Pfoser. Data semantics in location-based services. *Journal on Data Semantics*, 3:168–195, 2005. 17, 27, 28

[UG04] M. Uschold and M. Grunninger. Ontologies and semantics for seamless connectivity. *ACM SIGMOD Record*, 33(4):58–64, 2004. 19, 24

[vBFA05] A.H. van Bunningen, L. Feng, and P.M.G. Apers. Context for ubiquitous data management. In *proceedings of International Workshop on Ubiquitous Data Management (UDM2005)*, pages 17–24, 2005. 5, 19, 21, 22, 65, 97, 99

[VHPA06] Le-Hung Vu, M. Hauswirth, F. Porto, and K. Aberer. A search engine for qos-enabled discovery of semantic web services. *Journal of Business Process Integration and Management,*, 1(4), 2006. 175

[VJBCS97] P.R.S. Visser, D.M.R. Jones, T.J.M. Bench-Capon, and M.J.R. Shave. Assessing heterogeneity by classifying ontology mismatches. In *proceedings of AAAI'97 Spring Symposium on Ontological Engineering*, 1997. 86

[vSPK04] M. van Setten, S. Pokraev, and J. Koolwaaij. Context-aware recommendations in the mobile tourist application compass. In *proceedings of International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'04)*, pages 235–244, 2004. 7

[Wei93] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993. 1

[WGZP04] X. Wang, T. Gu, D. Zhang, and H. Pung. Ontology based context modeling and reasoning using owl. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communication (PerCom'04)*, 2004. 23, 97, 98, 100

[WHFG92] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transaction on Information systems*, 10(1):91–102, 1992. 2, 6, 16, 25

[WIY01] D.H. Widyantora, T.R. Ioerger, and J. Yen. Learning user interest dynamics with a three-descriptor representation. *Journal of the American Society of Information Science and Technology (JASIST)*, 52(3):212–225, 2001. 26

[XS05] R. Xie and R. Shibasaki. A unified spatiotemporal schema for representing and querying moving features. *SIGMOD Record*, 34(1):45–50, 2005. 7

[YAJS05] S. Yu, L. Al-Jadir, and S. Spaccapietra. Matching user's semantics with data semantics in location-based services. In *proceedings of 1st Workshop on Semantics in mobile Environments (SME 2005)*, 2005. 19, 22, 24, 119

[YSCA04] S. Yu, S. Spaccapietra, N. Cullot, and M-A. Aufaure. User profiles in location-based services: Make humans more nomadic and personalized. In *proceedings of IASTED International Conference on Databases and Applications*, 2004. 7, 26, 33

[ZG04] F. Zhao and L.J. Guibas. *Wireless sensor networks: an information processing approach*. Morgan Kaufman Publisher, 2004. 1, 21, 22

[ZGL03] V. Zeimpekis, G.M. Giaglis, and G. Lekeko. A taxonomy of indoor and outdoor positioning techniques for mobile location services. *ACM SIGecom Exchanges*, 3(4):19–27, 2003. 2

[ZZP$^+$03] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *Proceedings of the 2003 ACM SIGMOD international Conference on Management of Data (SIGMOD'03)*, pages 443–454, 2003. 7

Shijun YU

Database Lab, I&C, EPFL
CH-1015, Lausanne
Tel: +41 21 693 6706

Birthdate: March 13, 1976
Nationality: Chinese
e-mail : `shijun.yu@epfl.ch`

Education

2002 - 2007	**Swiss Federal Institute of Technology, Lausanne (EPFL)**, Ph.D of Computer Science.
2001 - 2002	**Swiss Federal Institute of Technology, Lausanne (EPFL)**, Postgraduate of Computer Science.
1994 - 1998	**Northern Jiaotong University, Beijing, China**, BS of Computer Science.

Professional Experiences

Sep.2002 ∼ Dec.2007 **Swiss Federal Institute of Technology, Lausanne (EPFL)**
Research and teaching assistant at Database Lab, School of Computer and Communication Science. *Responsibilities* included conducting research, lecturing and supervising projects for master-level students, giving tutorials in exercises and projects for Relational database course.

Jul.1999 ∼ Oct.2001 **Beijing Hope Computer Company, Beijing, China, affiliated with CAS (Chinese Academy of Sciences)**
Designed and implemented the government-oriented, e-Document Management System (eDMS). Architecture included: Self-contained database management, remote processing of handwriting files and digital signatures, customization of e-Document's delivery flow, cryptography and de-cryptography, and software distribution. Responsible for the project infrastructure, analysis of customer requirements, debug coordination between different modules, and partial documentation of the user manual. It has currently evolved as one of the most influential products in the company and already passed through the identification of ISO9000 quality standards. Its customers range from enterprises, government and educational institutions in six provinces in China.

Sep.1998 ∼ May1999 **Botong Technology Co. Limited, Tianjin, China**
Developed the MS/SQL-based Database management and designed the user Interfaces in Water-fee Revenue Information System of City 'Tianjin'. Architecture included: MS/SQL Server 7.0 database, Windows 2000 server, Windows terminals. System served for millions of users (city population: 8.5 million) and functioned from citizens daily usage to industrial consumption. Responsible for designing and managing data collecting and report production of daily consumption. It passed the final review of experts and continues to support the water-expense management and calculation for the city of Tianjin.

Professional Skills

• **Databases:** Design, development, and administration: Oracle9i, Oracle 10g, MS SQL 6.0, DBMain (teaching tools). Familiar with Conceptual modeling, ER and UML.
• **Programming Languages**: Design, development and debug experiences: Java (2 years+), VB (4 years+), PL/SQL (4 years+), C (1 year+), JavaScript (1 year), ASP (1 year+), XML (1 year), Python (beginning).
• **Web services and techniques:** UDDI/SOAP, OWL, OWL-S, WSMO, XML/XPath, Ontologies.

Publications

• Shijun Yu, Lina Al-Jadir, Stefano Spaccapietra. *Matching User's Semantics with Data Semantics in Location-Based Services.* 1st Workshop on Semantics in mobile Environments (SME 2005), Ayia Napa, 9 May 2005.
• Stefano Spaccapietra, Lina Al-Jadir, Shijun Yu. *Somebody, sometime, somewhere.* International Workshop on Ubiquitous Data Management (UDM2005), in conjunction with IEEE ICDE 2005, Tokyo, 4 Apr 2005.
• Shijun Yu, Stefano Spaccapietra, Nadine Cullot, Marie-Aude Aufaure. *User Profiles in Location-based Services: Make Humans More Nomadic and Personalized.* IASTED International Conference on Databases and Applications, Innsbruck, Austria, February 17-19,2004.
• Shijun Yu, Marie-Aude Aufaure, Nadine Cullot, Stefano Spaccapietra. *A Collaborative Framework for Location-Based Services.* CAiSE 2003, Klagenfurt/Velden, Austria, 16 - 20 June, 2003.
• Shijun Yu, Marie-Aude Aufaure, Nadine Cullot, Stefano Spaccapietra. *Location-Based Spatial Modelling Using Ontology.* 6th AGILE Conference, Lyon, France, April 24-26, 2003.

Personal Profile

Languages:	Chinese (native), English (fluent), French (fair).
Interests:	Traveling, photography, sports (e.g., badminton, hiking, yoga), cooking, flower-gardening, music.