Un modèle générique pour la capture de préférences dans les bases de données à base ontologique

Dilek Tapucu

Ecole Nationale Supérieure de Mécanique et d'Aérotechnique

Ecole Doctorale des Sciences et l'Ingénierie pour l'Information

THESE

pour l'obtention du grade de

DOCTEUR DE L'ECOLE NATIONALE SUPERIEURE DE MECANIQUE ET D'AEROTECHNIQUE

(Diplôme National — Arrêté du 7 Août 2006)

Secteur de Recherche : INFORMATIQUE et APPLICATIONS

Présentée par :

Dilek TAPUCU

****

A generic model for handling preferences in Ontology Based Databases

****

Directeurs de Thèse

Yamine AIT-AMEUR et Murat Osman UNALIR

****

Soutenu le 02 Juillet 2010
Devant la Commission d'Examen

****

JURY

Président :	Oğuz DIKENELLI	Professeur, Ege University, Izmir
Rapporteurs :	Djamal BENSLIMAN	Professeur, Université Claude Bernard, Lyon
	Ismail Sıtkı AYTAÇ	Professeur, Izmir Yüksek Teknoloji Enstitüsü, Izmir
Examinateurs :		
	Chihab HANACHI	Professeur, Université de Toulouse 1, Toulouse
	Yamine AIT-AMEUR	Professeur, ENSMA, Futuroscope
	Murat Osman UNALIR	Asst. Prof, Ege University, Izmir

Thanks

First and foremost, I would like to thank my supervisors - Prof. YAMINE AIT AMEUR. It is no more than luck to have his to guide me through my Ph.D research. He is not only knowledgeable but precise, always being able to find my problems in the first time and teach me how to correct them. Under the supervision of him, I have learned "how to work the plan, and plan the work". I deeply believe that I can benefit from it in my whole life. - Ass. Prof. MURAT OSMAN ÜNALIR. He inspired me to discover my research interests, supported my gradual change of topic. He has spent a lot of time, informing me the latent problems and advising me the right methods in conducting research.

Thanks to all at IYTE and EGE, in particular my colleagues in the laboratory LISI, for the cooperative and stimulating working atmosphere. And thanks to LADJEL BELLATRECHE and STEPHANE JEAN, who were always open for discussions and provided much feedback with their experience and knowledge.

I am grateful to GUY PIERRA and SITKI AYTAÇ, always optimistic and positive, with a sharp eye for improvements and real-world insight.

This thesis is dedicated to my family who have always provided me with the highest degree of love and support.

My Family.

Table of Contents

Part I

INTRODUCTION

Chapter 1

Introduction

Contents

Abstract. In this thesis, we propose a sharable, formal and generic model to represent user's preferences. The model gathers several preference types proposed in the Database and Semantic Web communities. The novelty of our approach is that the defined preferences are attached to the ontologies, which describe the semantic of the data manipulated by the applications. Moreover, the proposed model offers a persistence mechanism and a dedicated language. It is implemented by using an extended Ontology Based Databases (OBDBs) system in order to take preferences into account. The implemented preference model is formally defined by using the `EXPRESS` data modeling language, which ensures a non-ambiguous definition and the approach is illustrated through a case study in the tourism domain.

1 Research Context

The Web has grown from a tool for communication into an indispensable form of communication. Although essentially developed to facilitate knowledge management, the reuse of information on the Web is limited. The lack of reuse is the result of data hidden inside relational databases: closed systems with a rigid schema structure, lack of universal, reusable identifiers, and lack of expressive and extensible schemas. The Semantic Web improves the Web infrastructure with formal semantics and interlinked data. It enables flexible, reusable, and open knowledge management systems.

The move towards open and interlinked data on the Web and the Semantic Web results in more open systems. In contrast to traditional database-driven applications, open systems liberate the data that they operate on. Sources are decentralised, data can be semi-structured with arbitrary vocabulary and contributions can be published anywhere. Opening up existing applications and their data would improve knowledge management but raises challenges. These challenges are:

- programmatic problems about the access and manipulation of linked data over the web,
- visualization and manipulation of the information graph,
- guidance of user provided content,
- finding relevant data in distributed sources.

Our work mainly addresses the last two questions. In this context, the interrogation, exchange and integration of data, contained in databases have become critical issues. At the heart of these problems there is the need to clarify the semantics in databases. To solve the first problem, the solution is to embed ontologies within databases. This approach allows us to increase the database, by which ontologies describe the semantics of concepts they represent. This enrichment of classical databases has led to Ontology Based Databases (OBDBs) [Dehainsala et al., 2007a]. The semantics of the data contained in these databases are provided by the ontologies they retain. Database is containing both data and ontologies that describe data meanings are called OBDB [Alexaki et al., 2001], [Broekstra et al., 2002] and query languages [Prud'hommeaux and Seaborne, 2006], [Jean et al., 2005a], that have been associated, can manipulate both ontologies and ontology-based data. However, OBDB and ontology query languages don't assist users in finding relevant results to their queries.

Preferences express the sense of wishes and preference based search is a popular approach for helping users to find relevant items. Users would like to find the best match between their wishes and the reality. Preferred terms presumably require less mental effort to process and reduce the energy expended in the interactive information-seeking process. Modeling preferences is difficult, because human preferences are complex, multiple, heterogeneous, changing, and even contradictory. Moreover, they are hard to evaluate and according to the user's goals and his/her current task, they should be evaluated in the context they have been expressed.

Capturing and exploiting user's preferences have been proposed as a solution to this problem in many domains, including database systems [Kießling and Kostler, 2000; Kießling, 2002; Chomicki, 2003; Agrawal and Wimmers, 2000; Koutrika and Ioannidis, 2004; Viappiani et al., 2006; Das et al., 2006], Data Warehouse [Bellatreche et al., 2005], the Semantic Web [Siberski et al., 2006; Gurský et al.,

2008; Toninelli et al., 2008], Information Retrieval [Daoud et al., 2007] and Human Computer Interaction [Cherniack et al., 2003]. Although preferences are defined by using an ontology in some approaches, most of the previously cited work, and particularly in the Database domain defined the preferences and their model according to the logical model underlying the targeted system. The use of the preferences requires having knowledge of this logical model. Within most existing information systems, even the notion of preference has been integrated in various application domains. However, preferences are not explicitly modeled. They are often encoded with difficulty and disseminated throughout the applications that exploit these information systems. Therefore, they can not be shared and must be defined and updated for each application. This is a burden for users, and yields to another layer of heterogeneous modeling.

2 Our Proposal

The area of semantics for functional specifications has already been well understood for quite a long time. In contrast, for non-functional specifications, there exists no commonly accepted understanding of what constitutes a definition of semantics. Various specialized approaches can be found in literature (e.g., [Staehli et al., 1995], [Sabata et al., 1997]), but they are either incomplete or domain specific. Preferences are known as non-functional properties of information systems (e.g. security, quality).

Rather than extending a specific ontology model, the research presented in this thesis consists in introducing a side model to describe the non-functional concepts together with the ontology model inside an Ontology Based Database. The advantage of this approach is the possibility to adapt non-functional descriptions to any ontology model keeping the definition of this ontology model unchanged. We particularly study the notion of preference in this context. Technically, this extension is possible, only if the meta-model, that allow us to describe the ontology model, can be manipulated.

The main contributions of our work are the following:

- a sharable and formal model of preferences is defined. This model includes preference constructors of other models. It is formally defined using `EXPRESS` language,
- a link between the proposed preference model and ontology model is defined. It allows to link preferences to class and property of a ontology,
- an extension of OBDB with the proposed preference model is defined. It is realised through the manipulation of the meta-model level,
- an extension of ontology query languages with a `PREFERRING` operator is proposed. It allows to express preference-based queries.

3 Thesis Structure

This thesis is organized in three parts. The first part comprises research context and describes the motivations of our proposal.

The second part concerns the context of the study. The purposes of this section are to present relevant studies to our research and to explain concepts used throughout the study. This part is composed of the following chapters;

Chapter 2 describes required background information and results of other relevant studies. Preference and personalization concepts are summarized in the context of Database, Semantic Web, and other research areas. Query operators, which were developed for implementation of preference-based query, are also investigated in this chapter.

Chapter 3 introduces the `EXPRESS` data modeling language that is used to establish Preference Model. This language combines ideas from the entity-attribute relationship family of modeling languages with object modeling ideas of the late 1980s. The major advantage of this language is its capability to describe structural, descriptive and procedural concepts in a common data model and semantics.

Chapter 4 presents the motivation of Ontology-Based Databases. It details a model of architecture for OBDB called OntoDB that was used in this study. Moreover, it introduces the OntoQL an ontology query language than has been associated to the ontoDB OBDB.

From the requirements defined in the second part, the third part describes the proposed model. Additionally, the `PREFERRING` operator that was developed for preference-based queries is explained and its implementation is illustrated in this part. This part is composed of the following chapters;

Chapter 5 presents the Preference Model which was developed to address the research problem. First, Resource Definition is explained. The ontology's instances are taken into account by referring to their corresponding ontology's entities. Next, Preference Model, that was established by the collection of the different preferences is defined. The generic characteristic of this model is provided by the capability to define a relationship with any ontology of a given domain. Finally, Preference Link, which was constituted by the Resource Definition and Preference Model is represented.

Chapter 6 demonstrates the integration of the Preference Model that was explained in Chapter 5 into the architecture of OntoDB. Functions of the OntoQL language are used to carry out the mentioned integration. How OntoQL Query Language was extended with the `PREFERRING` operator for preference-based querying is explained in this chapter.

Chapter 7 explains the approach of Preference Model and implementation of the preference-based query on OntoDB architecture, by using two applied examples. These examples were given over a tourism scenario.

Lastly, we conclude this thesis by an overall conclusion and presents interesting future work opened by the work done. This thesis has three appendixes. Appendix A shows Preference Model with EPRESS code, Appendix B gives UML diagrams of Preference Model. Finally Appendix C provides the complete syntax of the OntoQL language.

Part II

CONTEXT OF THE STUDY

Chapter 2

Preference Driven Personalization Approaches

Contents

Abstract. Both the Web and Semantic Web can be seen as environments for knowledge management, and both have been influenced by the vision of the personalization for personal knowledge management. This chapter explains the required background information for preference, personalization concepts, and summarizes different research areas about this context. The chapter finishes with comparisons between preference definition approaches in Database, Semantic Web, Data Warehouse and a list of missing points in the literature in the context of Ontology Based Databases.

1 Introduction

The rapid growth and the wide adoption of internet technology made a huge amount of data managed by various information systems available. When searching over these disseminated data, users are often encountered by the numerous returned results in response to their requests. These results must often be sorted and filtered in order to identify the relevant information. Despite the fact that the "one size fits all" approach has shown its limitation in many applications, most information systems do not take the variety of user's needs and preferences into account.

Preferences represent the basic notion for any decision support activity. One of the principal tasks within a decision aiding process is to model preferences in such a way that it is possible to derive a final recommendation for the decision maker. The problem is that quite often the decision maker adopts preference statements in "natural language" which do not necessarily have a straightforward modeling. Within many domains including the Database Systems [Kiesling and Kostler, 2000], [Kiesling, 2002], [Chomicki, 2003], [Agrawal and Wimmers, 2000], [Koutrika and Ioannidis, 2004], [Viappiani et al., 2006], Data Warehouse [Bellatreche et al., 2005], Semantic Web [Siberski et al., 2006], [P. Gurský and Vaneková, 2008], [Toninelli et al., 2008], Information Retrieval [Daoud et al., 2007] and Human Computer Interaction [Cherniack et al., 2003] capturing and exploiting user's preferences have been proposed as a solution to this problem.

Preference-based queries can be used in order to increase the expressivity of queries, helping users to describe their wishes more accurately and interests and retrieve efficiently optimal matches according to the user preferences discarding the rest.

Personalization is a process of adapting and filtering an information flow. It gives feedback interactively at real-time, according to the individual's preferences [Al and H., 2003]. Personalization technologies gained significance in the 90's, with the boost of large-scale computing networks which enabled the deployment of services to massive, heterogeneous, and less predictable end consumer audiences. As the number of services and the volume of content (text and multimedia; public, commercial and personal) in these networks keeps growing, personalization is more than ever a critical enabler in helping consumers to manage capacity and complexity, and help vendors (content providers, managers, brokers, distributors, technology providers) to reach their target users.

According to Adomavicius and Tuzhilin [Adomavicius and Tuzhilin,] the goals of a personalization system are to:

- provide precise and related content, based on each user's preferences,
- provide user satisfaction, by understanding the user needs and trying to meet them successfully,
- determine user's preferences with minimal participation of them,
- recommend products in real time, so that users can react quickly. And this increases the user loyalty and encourages them to re-use the offered services.

This chapter describes a personalization system that has been developed in this perspective and addresses the state of the art. Section 2 gives the preference concept with definitions. Section 3 presents the personalization approaches in a generic way. In Section 4 preference based researches in the Database

research domain are overviewed and in Section 5 Semantic Web research area and preference concept are explained together. In Section 6 a brief summary of all research works is given. Finally, Section 7 concludes this chapter.

2 Concept of Preference

The notion of preference is becoming more and more ubiquitous in present day information systems. Preferences are primarily used to filter and personalize the information, which reaches the users of such systems. Although preferences have traditionally been studied in fields such as economic decision making, social choice theory, and operations research, they have nowadays found significant interest in computational fields such as Artificial Intelligence, Databases, and Human-computer interaction. This broadened scope of preferences leads to new types of preference models, new problems for applying preference structures, and new kinds of benefits.

There are many preference definitions. In Philosophical definition, "preferences are used to reason about values, desires, and duties" [Hansson, 2001]. In Mathematical Decision Theory, "preferences (often expressed as utilities) are used to model people's economic behavior" [Fishburn, 1988]. For Databases concept, "preferences help in reducing the amount of information returned in response to user queries" [Lacroix and Lavency, 1987] and for Artificial Intelligence, "preference relations serve to establish an intervention goal of an agent" [Dubois et al., 1998]. Although, all these definitions are separated from each other, they have also common values to affect decision making process.

Preference based systems allow finer-grained control over computation and new ways of interactivity, and therefore provide more satisfactory results and outcomes. Preference models provide a clean understanding, analysis and validation of heuristic knowledge used in existing systems such as heuristic orderings, dominance rules, and heuristic rules.

Preference modeling is a popular approach for helping consumers to find their desired items. Classical models are utility functions that map the possible outcomes of the decisions to numeric values and thus allow the comparison and sorting of those outcomes. Explicit preference modeling provides a declarative way to choose among alternatives, like solutions of problems to find answers to database queries, decisions of a computational agent, plans of a robot, and so on. Weak preference orders is another model that describes which outcome is the least preferred. This model shows user's negative preferences. Artificial intelligence researches bring new alternative application fields to these classic preference models.

3 Personalization

Personalization in the World Wide Web can be compared to creating individual views on Web data according to the special interests, needs, requirements, goals, access-context, etc. of the current beholder. In general, the goal of personalization is to accommodate user preferences, to improve the efficiency of the interaction with users, and to make complex systems more usable [Fischer, 2001].

Query personalization is another process of dynamically enhancing a query with related user prefer-

ences stored in a user profile, with the purpose of providing personalized results.

From the utility perspective, personalization is important when significant differences between users are observed. An important form of personalization is interface customization. Interface customization is to customize user's online experience by adapting the user interface to suit their preferences.

Personalization techniques make it possible to change the structure and content delivered to the users in order to match the needs and preferences of users based on a user profile, which is stored and updated dynamically.

According to Jorstad and Thanh [Jorstad et al., 2006] "Personalization is the process where services are adapted to fit each individual user's requirements (needs and preferences)", and personalization entails the following steps:

- collecting information about the user to build services preference profile. These preferences could be gathered by subscription process or user-rating mechanism,
- storing and keeping regular updates for this information,
- recommending personalized services to a targeted user.

Personalization is of type explicit or implicit [Klusch et al., 2003]. Explicit personalization calls for a direct participation of users in the adjustment of applications. Users clearly indicate the information that needs to be treated or discarded. Implicit personalization does not call for any user involvement and can be built upon learning strategies that track users' behaviors. Personalization is dependent on the features of the environment in which it happens. These features can be related to computing resources (e.g., fixed device, mobile device), time periods (e.g., in the afternoon, in the morning), and physical locations (e.g., meeting room, cafeteria). The gathering and refinement of an environment's features permit defining its context. Context is the information that characterizes the interaction between humans, applications, and the surrounding environment [Brézillon, 2003]. There are many personalization techniques, three most useful of them are described below.

3.1 User Profiling

User profile construction is an important component of any personalization system. The concept of a user profile usually refers to a set of preferences, information, rules and settings that are used by a product or service to deliver customized capabilities to the user. The term user profiling is used to refer to a software module that acquires personal data of a user, processes these data to obtain additional information, and uses it to modify either content aspects or navigation capabilities of web pages. The more common techniques are explicit and implicit profiling [Buono et al., 2001].

- Explicit profiling: each user is asked to fill in a form when visiting the web site; this method has the advantage of letting users specify their interests directly.
- Implicit profiling: the user's behavior is tracked automatically by the system. This method is generally transparent to the user. Often, user registration is saved in what is called a cookie that

is kept within the browser and updated at each visit. Behavior information is generally stored in a log file.

Personalization exploits profiles of the users interacting with the system. An individual user profile includes assumptions about their knowledge, beliefs, goals, preferences, interests, misconceptions, plans, tasks, abilities, work context, etc [Kobsa, 2001]. The forms that a user profile may take are as varied as the purposes for which user models are formed. User models may seek to describe: the cognitive processes that underlie the user's actions; the differences between the user's skills and expert skills; the user's behavioral patterns or preferences; or the user's characteristics. With reference to [Kobsa, 2001], author suggests distinguishing adaptation to user data, usage data, and environment data. User data comprise the adaptation target, various characteristics of the users. Usage data comprise data about user interaction with the systems that cannot be resolved to user characteristics. Environment data comprise all aspects of the user environment that are not related to the users themselves (e.g., user location and the user platform). Brusilovsky [Brusilovsky, 2001] defines uses data by dividing it into:

- user characteristics (user's goals/tasks, knowledge, background, experience, and preferences),
- user interests (long-term interests such as a passion and short-term interest such as search goal),
- user's individual traits (e.g. personality factors, cognitive factors, and learning styles.).
- user groups and stereotypes model. They are the representation of relevant common characteristics of users pertaining to specific user subgroups of the application system [Kobsa, 2001].

Profiles promise to ease the conflict between the benefits of common technology deployments versus diverse social and cultural demands, and variations in individual physical and cognitive abilities and preferences. To achieve effective personalization, profiles should distinguish between long-term and short-term interests and include a model of the user's context, i.e., the task in which the user is currently engaged and the environment in which they are situated. Several systems have attempted to provide personalized search that are tailored based upon user profiles.

3.2 Search Engines

Information on the Web is huge and growing rapidly. An effective search engine is an important means for users to find the desired information from billions of Web pages. Users tend to issue short queries when searching, resulting in tremendous ambiguity about their informational goals. In order to achieve this web search engines are beginning to offer personalization capabilities to users. Personalization of search results is very important to the future success of any search engine.

Some search engine Web sites offer personalization and others provide customization, or both. In personalization, a user created profile decides the personalizable solution whereas in customization, the user is allowed to select from a predefined set of solutions. Though personalization seems to be a simple concept to understand, it is not easy to implement. Problems include, personalization solution which means that the solution which works for one individual will not necessarily be a personalization solution

for another. Thus, it's difficult to provide a personalization solution that is complete for each user. Search engine Web sites can provide a generic set of personalizable features.

Currently Web sites rely heavily on the user's inputs for a personalization solution. [Mobasher et al., 2000] propose a general architecture for Automatic Web personalization. The architecture attempts to automate the personalization process by tracking the user's preference from the Web server logs. Perkowitz and Etzioni [Weld et al., 2003] addressed this problem by designing an adaptive Web site that relies heavily on the user's navigation pattern and tries to anticipate the user's need based on his past navigation history. [Damiani et al., 2001] describe the WBI (Web Browser Intelligence) architecture for personalizing Web sites. The previous two approaches rely on a server for the personalization process, but the WBI architecture can be used on the client side, middleware or the server side. Few studies have surveyed the nature and extent of search engine Web sites that include personalization features. There has been a growing interest in making the personalization process completely automated. Presently the Web sites rely heavily on the user's inputs for presenting them a personalizable solution. [Manber et al., 2000] discuss the applications for personalization that exists on the Web and the use of profiles in the personalization process.

Adapting a search engine to cater for specific users and queries is an important research problem and has many applications. In general, there are two aspects of search engine adaptation that need to be addressed. The first aspect is query specific adaptation; that is, how to return the best results for a query from the underlying search engines that have different coverage and focuses. The second aspect is user specific adaptation that aims to meet the diversified preferences of different users in the search results.

3.3 Recommender Systems

Recommender systems are one of the most popular applications of personalisation techniques [Adomavicius and Tuzhilin,]. Basically, the aim of the recommender system is to suggest interesting items to the users' automatically, based on their preferences. Many e-commerce sites have successfully utilized different types of recommending systems as a means to offer personalised customer service and to improve the online shopping experience [Prassas, 2001].

The collaborative (social) recommender systems are the most well known type of recommender systems. These systems aggregate data about customers' purchasing habits or preferences, and make recommendations to other users based on similarity in overall purchasing patterns. For example, in the Ringo music recommender system [U. Shardanand, 1995], users express their musical preferences by rating various artists and albums, and get suggestions of groups and recordings that have similar features.

The content-based recommender systems are classifier systems derived from machine learning research. These systems use supervised machine learning to induce a classifier, that can discriminate between items likely to be of interest to the user and those likely to be uninteresting.

The personal logic recommender systems offer a dialog, that effectively walks the user down a discrimination tree of product features. Others have adapted quantitative decision support tools for this task [H. K. Bhargava and Herrick, 1997].

4 Personalization in Databases

Handling preferences in the database domain has been addressed in many research studies ([Kiesling and Kostler, 2000], [Kiesling, 2002],[Chomicki, 2003], [Agrawal and Wimmers, 2000], [Koutrika and Ioannidis, 2004], [Viappiani et al., 2006]). Preferences in this context are defined on the logical model level of the database, specifically on the column values of the tables. According to the type of used metric, two different ways of expressing preferences have been proposed: qualitative and quantitative approaches.

Qualitative approaches [Kiesling, 2002],[Chomicki, 2003] allow users to define (relative) preferences between tuples. The preferences are defined on the content and define a binary relation between tuples [Chomicki, 2003]. For example, if we consider two tuples t1 and t2 , the expression `t1 > t2` means that the user prefers the tuple `t1` rather than `t2` . Kießling and Kostler follow a qualitative approach as well, named constructor approach. The preferences are expressed by a strict partial order and are formally described by first order logical formulas [Kiesling, 2002]. The defined constructors are integrated within the Preference SQL relational language [Kiesling and Kostler, 2000]. For instance, the constructor `Highest(c)` is used to express that for 2 tuples `t1` and `t2` , we prefer the tuple having the higher value for the column c. This approach is referred as the BMO (Best Match Only) query model and is identical to the `winnow` operator defined by Chomicki [Chomicki, 2003].

Quantitative approaches on the other hand allow users to define scoring functions to compute a numeric score or an absolute preference for each tuple [Agrawal and Wimmers, 2000], [Koutrika and Ioannidis, 2004]. The results are sorted according to this score. In this context, Agrawal and Wimmers define preferences by introducing a preferred value for each column in the database's tables [Agrawal and Wimmers, 2000]. For instance, let us consider the table Hotel defined as `Hotel(name, priceMin, priceMax)`. The preference < `*,40,80` > indicates that preferred hotels are those having room price between 40 and 80. This preference is then used to compute a score between 0 and 1 for each hotel. Koutrika and Ioannidis introduce the notion of atomic preferences by specifying a set of pair < `condition, score` > where condition is a condition on the values of columns and score is the degree of interest between 0 and 1 of this condition [Koutrika and Ioannidis, 2004]. Atomic preferences can be combined and used to derive implicit preferences. For example, considering the same table Hotel, the expression < `Hotel.name = Sophitel, 0.8` > indicates that the interest degree of Sophitel Hotels is 0.8. Also the quantitative approaches propose a presentation based preference of user profiling. These preferences define an order relation between the tuples returned by a given query. In this case, two tuples are compared after accessing data sources. The comparison is based on a set of selected attributes. The presentation is expressed by the selection of a set of relevant attributes and by displaying the corresponding tuples in the tables and ranked according to their importance.

These approaches are strongly linked to the logical model of the database. Building a logical model for a database system is a fundamental problem in many database related applications, such as data integration [Giacomo and Lenzerini, 1995], knowledge representation, and data warehouse, etc. Logical model for a database system transforms each database system into a logical system with enriched semantics and enhanced reasoning mechanism. Operations in database systems, such as query processing, could be done based on the logical inference mechanism. Therefore, a good knowledge of this logical model is required for an efficient exploitation of these models.

4.1 Preference Formulas in Relational Queries

Database queries are often exploratory and users often find that their queries return too many answers, many of which are irrelevant. Existing studies either categorize or rank the results to help users locate interesting results. The success of both approaches depends on the utilization of user preferences. However, most existing studies assume that all users have the same user preferences, but in real life different users often have different preferences.

Firstly, Lacroix and Lavency [Lacroix and Lavency, 1987] addressed composability in showing, how multiple preference conditions could be combined and prioritized. As their extended query language inherits the same first order logic definition as the DRC (Domain Relational Calculus), their language has declarative semantics. The approach is limited in expressiveness, though. Each preference condition is evaluated in a Boolean manner: either there are answers that satisfy the query or there are not (and so the query is evaluated as without that preference).

Personalization of database queries is an increasingly important issue. User preferences can be embedded into database query languages in several different ways. To provide more effective search capabilities, query languages like SQL over relational databases have been extended to facilitate preference based retrieval algorithms. In this section, we briefly present various operators including winnow [Chomicki, 2003], Skyline [Borzsonyi et al., 2001] [Theobald et al., 2004] and Pareto [Viappiani et al., 2006] which are applied on the database tables' columns.

The Winnow operator is proposed by Chomicki for composing preference relations in the relational algebra. They introduced a general logical framework for preferences, as preference formulas.

The Skyline operator was introduced to the database context by applying the problem of finding the maxima of a set of points. Given several aspects for ranking data items, the skyline of a given data set is defined as that subset which contains exactly all "interesting" items.

Pareto and prioritized preference construction preserves strict partial orders, which instantly solves crucial well known problems for preference queries.

4.2 Preference Based SQL

In personalized database applications a cooperative query model is needed which supplements the exact-match query of SQL or XPath. Kießling [Kiesling, 2002] has taken an algebraic approach to construct a rich preference query language as an extension to SQL, that is called `Preference SQL` . `Preference SQL` allows users to write best-match queries by composing their preference criteria, via the preference operators. Preference SQL has been on the market since 1999, and is used in several commercial ventures. The system compiles preference queries into SQL for evaluation. In [H. Stefan and Kiesling, 2003] Kießling and Koestler investigate further how to extend SQL and XPATH for the `Preference SQL` operators, and present rich examples of the types of queries that can be composed. A partial syntax of the extended query language is given below:

```
SELECT <projection-list>
 FROM <table-reference>
 WHERE <hard-conditions>
```

```
 PREFERRING <soft-conditions>
ORDER BY <attribute-list>
```

Using this syntax, the user can express their preferences as soft constraints and will receive tuples which best match those constraints. This approach is referred to as the *BMO* (Best Match Only) query model, in which a tuple will find its way into the final result set if there aren't other tuples which dominate it, i.e. better satisfies the preference constraints, do not exist.

Preference SQL introduces many new constructs. And how to realize them efficiently is a challenge. The current system translates queries into SQL. It would be hard to integrate the preference mechanisms within a relational engine, because of the extensive additions. Preference SQL has an operational semantics, but not a defined declarative semantics. In particular, composition of the preference operators can raise difficulties. The intended semantics is that the preference relation be a partial order, but certain compositions can violate this. Kießling proposes the concept of substitutable values (SVs) and SV relations to address sound composition of Preference SQL's Pareto and prioritized preferences.

5 Ontology Based Knowledge Personalization

Semantic-based techniques enable to infuse software systems with a more precise understanding of application-domain knowledge, and henceforth, provide better means to define user needs, preferences, and activities within or with regards to the system.

In this context, the Semantic Web [Gruber, 1993], [Berners-Lee et al., 2001] enables automated information access and use based on machine-processable semantics of data. It can be regarded as an extension of the existing Web, whose information is mostly human-readable. The Semantic Web allows for finer granularity of machine-readable information and offers mechanisms to reuse agreed-upon meaning. It simplifies knowledge discovery, reuse, and management by explicitly and formally representing information about online data sources.

Ontologies [Gruber and Olsen, 1994] are the backbone technology for the Semantic Web and - more generally - for the management of formalized knowledge in the context of distributed systems. They provide machine-processable semantics of data and information sources that can be communicated between different agents (software and people). In other words, information is made understandable for the computer, thus assisting people to search, extract, interpret and process information.

To provide a personalized environment to a user, a consistent domain ontology is important. The goal of ontology based knowledge search personalization is to tailor search results to a particular user based on that user's interests and preferences. Effective personalization of information access involves two important challenges: accurately identifying the user context and organizing the information in such a way that matches the particular context. Thanks to the semantic movement of the last years, a wide variety of both specific and more general ontologies exists today. This abundance of knowledge enriches our central domain ontology and gives possibility to make use of it.

5.1 Preferences in Semantic Web

The objective of the Semantic Web is a content-aware navigation of the resources. This means being able, by means of proper mechanisms, to identify those resources, that better satisfy the requests not only on the basis of descriptive keywords but also on the basis of knowledge. There is, in fact, a general agreement that the use of knowledge increases the precision of the answers. Such knowledge represents different things, such as information about the user, the user's intentions and the context. One of the key features that characterize the Semantic Web is that its answers are always personalized or adapted so as to meet specific requirements. It will not be the case that the answer to a query about 'book' will contain links to bookshops and links to travel agencies. This Web of knowledge is currently being built on top of the more traditional World Wide Web and requires the definition of proper languages and mechanisms.

The goal of personalization in the Semantic Web is to make the access to the right resources easier. This task entails two orthogonal processes: retrieval and presentation. Retrieval consists of finding or constructing the right resources when they are needed, either on demand or (as by the use of automatic updates) when the information arises in the network. Once the resources have been defined, they are presented in the most suitable way to the user with taking into account his/her own characteristics and preferences. To these aims, it is necessary to have a model of the user, that is, a representation of those characteristics according to which personalization will occur. It is also necessary to apply inference techniques which, depending on the task, might range from the basic ontology reasoning mechanisms supplied by Description Logics (like subsumption and classification) to the most various reasoning techniques developed in Artificial Intelligence. In this research area different models of preferences have been proposed in the literature [Siberski et al., 2006], [P. Gurský and Vaneková, 2008], [Toninelli et al., 2008].

The Local Preference Model is proposed by Gurský et al. in order to model complex user preferences [P. Gurský and Vaneková, 2008]. They consider that complex preferences reflect real life preferences more accurately. They use a fuzzy based approach for preference description. Firstly, nominal and ordinal attributes are used to define local preferences. Then, their combination with user's local preferences produces global preferences. For example, the global preference `good hotel(x)` can be defined by the combination of the two local preferences expression `good price(x)` and `good starRating(x)`.

Toninelli et al. introduce the Ontology Based Preference Model approach, by defining a meta model [Toninelli et al., 2008]. In this approach, value and priority preferences are specified. For example, to find high standard hotels, the quality of the service must be a priority.

Sieg et al. present an ontology based approach for personalising Web information access [Sieg et al., 2004]. The user interests are captured implicitly by a context, defined through the notion of ontological user profiles. This context model for a user is represented as an instance of reference domain ontology. The concepts of the ontology are annotated by interest scores derived and updated implicitly according to the user's behavior.

5.2 Preference-driven Query Processing in Semantic Web

The SPARQL query language allows queries over RDF Graphs using Triple Pattern Matching by introducing variables and binding the appropriate RDF resources to the variables. In a similar fashion to the

way that Kießling extended SQL to enable database querying with preferences, [Siberski et al., 2006] presents an extension to SPARQL to query ontological information with preferences. The fundamental idea here is similar; a new query element is introduced to allow the construction of preferences as soft constraints. The extended syntax of SPARQL is given below:

```
SELECT <projection-var-list>
FROM <ontology-reference>
WHERE <var-bindings>
FILTER <hard-conditions>
PREFERRING <soft-conditions>
ORDER BY <var-list>
```

In the `PREFERRING`, every filter operator supported by SPARQL can be used as well as two scoring operators, `HIGHEST/LOWEST` with similar semantics as Preference- SQL. Also, similarly to Preference SQL, two complex preference assembly methods are implemented, i.e. the Pareto operator for treating two preference operators as equally important and the Cascade operator to prioritize one preference operator over the other. Finally, in this work the BMO (Best Match Only) query model was adopted where a solution binding is a best match if there is no other solution binding dominating it (i.e., strictly preferred). Each solution binding competes against every other solution binding where a solution binding will find its way into the final result set if it is a "best match" under this definition.

6 Other Research Areas

This overview of the use of preferences in information system is not complete. Also the studies about Information Retrieval [Fuhr et al., 1999], Data Warehouse areas [Bellatreche et al., 2005] and Human-Computer Interaction Systems must be examined.

Human Interaction

User modeling started in the early 80's and human-computer interaction [Cherniack et al., 2003] in the 60's. In a computer assisted solution of corporate memory management, the interaction with users has to be studied. The two fields are extremely linked both historically and in their research objectives; in *human-computer interaction*, the human is quite often a user, whose model must be taken into account to improve the behavior of the system. In a *knowledge management* perspective, the user is part of the context, and the context is an important factor when knowledge is handled. Therefore, user modeling has a role to play in knowledge management solutions. On the other side, the problem of modeling humans and their cognitive activities will raise considerations that fall within the competence of knowledge representation and knowledge-based systems. Those two domains can complement each other.

Data Warehouse

There are few studies about the preferences in the data warehouse context compared to database and Semantic Web. In [Bellatreche et al., 2005] a personalization framework for OLAP queries is presented.

They give end user the possibility to specify her/his preferences (e.g., the presence of a given dimension of a data warehouse in the final result) and her/his visualisation constraint to display the result of an OLAP query. The visualisation constraint represents the size of device (PDA, mobile phone, etc.) used to display the result of a query. The authors present some issues on the impact of preferences on physical design of a data warehouse (data partitioning, index and materialized view selection). This work did not present query operator handling preferences.

7 Conclusion

In this chapter, previously published researches are reviewed in terms of personalization and preference concepts. The presented approaches are summarized in Table 2.1. They are classified according to five criteria. The first criterion indicates whether the considered approach is presented in the context of Database (DB), Semantic Web (SW) or Data Warehouse (DW). The second criterion indicates the followed approach (qualitative, quantitative, etc.). The third criterion indicates at what level (physical, logical, semantic) preferences are defined. The fourth criterion indicates whether there is a specific operator for querying with preferences. Finally, the last criterion indicates whether the considered approach offers the possibility to store physically the preferences model. As a result,

- personalization approach in the area of Ontology Based Database which is between the research area of Database and Semantic Web, was not investigated in literature. Also, there was no Preference Model or Preference Based Query implementation,
- there is no generalized model that gathers different preference types found in Databases and Semantic Web communities that were defined in other studies to establish a preference model,
- there exist not an established preference model used to develop a preference operator that will be used in preference based queries and
- it was also observed that preference based query was not carried out by using preference types that were defined on preference model.

These observations lead us to define a new Preference Model that can be attached to ontologies, stored in OBDB and exploited in ontology-based queries. To avoid ambiguities this models is formally defined with `EXPRESS` language presented in the next chapter.

Table 2.1: Preference definition approaches in Databases, Semantic Web and Data Warehouses domains

Author	Domain	Approach (DB/SW/DW)	Preferences	Query Operator Model Level	Storage of Preferences Model
Kiebling (2002-2003)	DB	Qualitative	Logical	Preference SQL	No
Chomicki (2003)	BD	Qualitative	Logical	Winnow	No
Agrawal-Wimmers (2000)	DB	Quantitative	Logical	No	No
Koutrica-Ionnidis (2006)	DB	Quantitative	Logical	No	No
Das et al. (2002-2003)	DB	Qualitative	Logical	No	No
—	**OBDB**	—	—	—	—
Siberski et al. (2006)	SW	Boolean-Scoring Preferences	Semantic	SPARQL, Clause Preferring	No
Sieg et al. (2007)	SW	Ontological User Profiles	Semantic	No	No
Gurský et al. (2008)	SW	Fuzzy based Ontology	Semantic	No	No
Tonielli et al. (2008)	SW	Middleware Meta Model	Semantic	No	No
Bellatreche et al. (2005) Mouloudi et al. (2006)	DW	Qualitative	Logical	No	No

Chapter 3

The EXPRESS Modeling Language

Contents

Abstract. In this chapter, the main features of the EXPRESS modeling language are described because this language is used in the realization of the Preference Model. EXPRESS is a standard data modeling language initially defined for product data. This language is similar to UML but provides a powerful and integrated constraints language.

1 Introduction

EXPRESS [IS010303.02, 1994], [Schenk and Wilson, 1994] is a data modeling language that combines ideas from the entity-attribute-relationship family of modeling languages with object modeling ideas of the late 1980s. It became an international standard (ISO 10303-11) in 1994 for use in engineering data exchange. It is formalized in the ISO Standard for the Exchange of Product model STEP (ISO 10303). STEP (STandard for the Exchange of Product model Data) is an international standard (ISO-10303, Industrial automation systems and integration Product data representation and exchange) for the computer interpretable representation and the exchange of product model data. The major advantage of this language is its capability to describe structural, descriptive and procedural knowledge in a common data model and semantics.

EXPRESS contains object oriented and procedural concepts as well as data base concepts. It enables the complete and non-ambiguous description of a mainly static product model. This language specifies an information domain in terms of entities, i.e. classes of objects sharing common properties which are represented by associated attributes and constraints. In EXPRESS , constraints are written using a mixture of declarative and procedural language elements. As in object models, an EXPRESS entity instance is considered to have an identity distinct from its modeled attributes and properties. That is, EXPRESS does not consider any attribute value or the set of attribute values to denote the entity instance. An entity instance is considered to be an object, which is partly represented by the modeled attributes, and has an unmodeled unique identifier.

EXPRESS is similar to programming languages such as PASCAL. Within a SCHEMA, various data types can be defined together with structural constraints and algorithmic rules. A main feature of EXPRESS is the possibility to formally validate a population of data types - this is to check for all the structural and algorithmic rules. Table 3.1, consider the Family EXPRESS schema presented above.

```
SCHEMA Family;
 ENTITY Person
  ABSTRACT SUPERTYPE OF (ONEOF (Male, Female));
   name:  STRING;
   mother:  OPTIONAL Female;
   father:  OPTIONAL Male;
END ENTITY;
 ENTITY Female
   SUBTYPE OF (Person);
 END ENTITY;
 ENTITY Male
   SUBTYPE OF (Person);
 END ENTITY;
END_SCHEMA;
```

Table 3.1: Family EXPRESS schema

It contains a super type entity Person with the two subtypes Male and Female. Since Person is declared to be ABSTRACT only occurrences of either (ONEOF) the subtype Male or Female can exist.

Every occurrence of a person has a mandatory name attribute and optionally attributes mother and father. There is a fixed style of reading for attributes of some entity type, a Female can play the role of mother and a Male can play the role of father for a Person.

Table 3.2 shows the entity B with three attributes: a real, a list of strings and a relationship with another entity A which has only one integer attribute. att_1 is an inverse attribute of entity A, corresponding to the inverse link defined by attribute att_3 in entity B.

Entities may have instances. Each instance is identified by an OID (Object IDentifier: # i). It is characterized by name of the class instantiated. An example of the model extension associated to the previous entity definitions is shown in the same Table. The #2 instance of the entity B, where att_1 evaluates to 4.0, att_2 is the list ('hello', 'bye') and att_3 points the particular instance #1 of the Entity A where its att_A attribute evaluates to 3.

Entity Definition

ENTITY A
  att_A(?):INTEGER;
  INVERSE;
  att_1:B FOR att_3;
END ENTITY;
ENTITY B
  att_1:REAL;
  att_2: LIST [0:?] OF STRING;
  att_3:A;
END ENTITY;

Entity Instance
#1=A(3); #2=B(4.0,('HELLO','BYE'), #1);

Table 3.2: Entity Definition and Instantiation in EXPRESS

An EXPRESS data model can be defined in two ways, textually and graphically. For formal verification and as input for tools such as SDAI the textual representation within an ASCII file is the most important one. The graphical representation called EXPRESS-G on the other hand is often more suitable for human use such as explanation and tutorials (e.g. Figure 3.1).

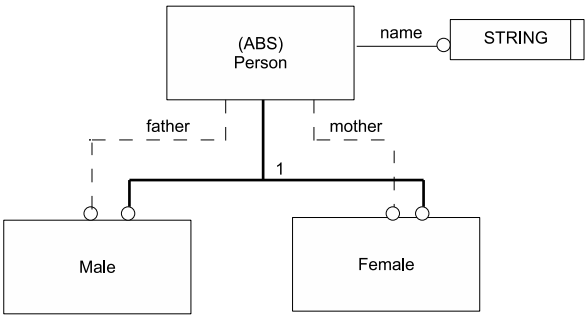


Figure 3.1: EXPRESS-G notation for Family example.

This chapter gives the main features of the EXPRESS modeling language. In the following, graphical representation of EXPRESS is described first. Then some background information on EXPRESS is given. Finally, the reason of EXPRESS usage in this thesis is explained. At the end, some conclusions are drawn and an outlook is given.

2 **EXPRESS** Building Blocks

Th building blocks of EXPRESS are entities, attributes, type declarations and hierarchies of inheritance and they are represented by using EXPRESS-G . EXPRESS-G is a diagrammatic modeling notation for the purpose of an object oriented information modeling. This notation is based on the standardized EXPRESS-G notation, which is itself described in ISO 10303-11 (Industrial Automation Systems Product Data Representation and Exchange Part 11 Description Methods: The EXPRESS Language Reference Manual). An introduction to EXPRESS and EXPRESS-G can be found in [Schenck and Wilson 1994].

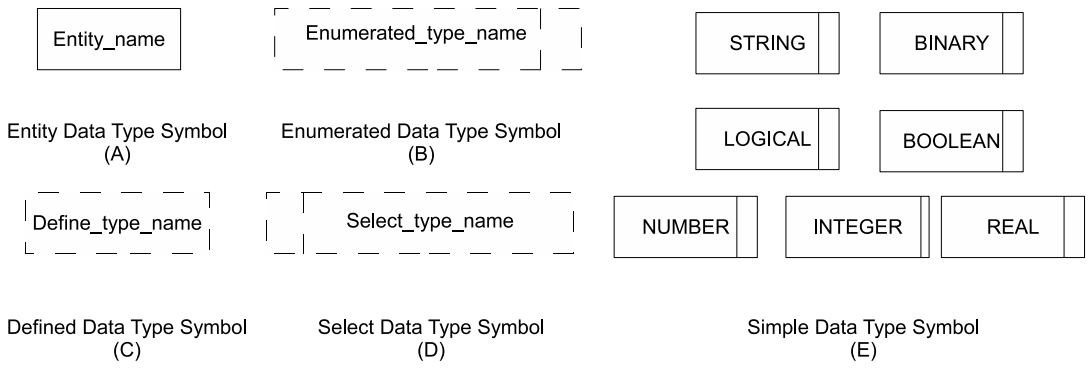


Figure 3.2: Data type symbols of the EXPRESS-G notation [Schenk and Wilson, 1994].

Entity Data Type (Figure 3.2-A):

The primary EXPRESS concept is the entity type, which models a domain of conceptual or real-world objects and the collection of information units that describe them. In the diagrammatic modeling notation, an entity is shown as a rectangular box in Figure 3.2-A, the entity name is written inside the box.

Enumeration Data Type (Figure 3.2-B):

A data type enumeration expresses the existence of a range of values belonging to this attribute. Only one of the enumerated values may be chosen. Figure 3.2-B depicts the symbol with a dashed box having a vertical bar on the right side of the box.

Def ned Data Type (Figure 3.2-C):

EXPRESS supports defined types (Figure 3.2-C), which are new data types defined by the modeler to be represented by values of any of the other data types. E.g. it is possible to define the data type positive which is of type integer with a value > 0.

Select Data Type (Figure 3.2-D):

Selects define a choice or an alternative between different options. Most commonly used are selects between different entity_types. More rarely are selects which include defined types. In the case that an enumeration type is declared to be extensible it can be extended in other schemas. Like in Figure 3.2-D the name of the data type is written within a dashed box having a vertical bar on the left side of the box.

Simple Data Type (Figure 3.2-E):

- **String:** This is the most often used simple type. EXPRESS strings can be of any length and can contain any character (ISO 10646/Unicode).
- **Binary:** This data type is only very rarely used. It covers a number of bits (not bytes). For some implementations the size is limited to 32 bit.
- **Logical:** Similar to the Boolean data type a logical has the possible values TRUE and FALSE and in addition UNKNOWN.
- **Boolean:** With the Boolean values TRUE and FALSE.
- **Number:** The number data type is a supertype of both, integer and real. Most implementations take uses a double type to represent a real_type, even if the actual value is an integer.
- **Integer:** EXPRESS integers can have in principle any length, but most implementations restricted them to a signed 32 bit value.
- **Real:** Ideally an EXPRESS real value is unlimited in accuracy and size. But in practice a real value is represented by a floating point value of type double.
- **Aggregation data type:** The possible kinds of aggregation_types are SET, BAG, LIST and ARRAY. While SET and BAG are unordered, LIST and ARRAY are ordered. A BAG may contain a particular value more than once, this is not allowed for SET. An ARRAY is the only aggregate which may contain unset members. This is not possible for SET, LIST, BAG. The members of an aggregate may be of any other data type (Figure 3.2-E).

A few general things are to be mentioned for data types.

- Constructed data types can be defined within an EXPRESS schema. They are mainly used to define entities, and to specify the type of entity attributes and aggregate members.
- Data types can be used in a recursive way to build up more and more complex data types. E.g. it is possible to define a LIST of an ARRAY of a SELECT of either some entities or other data types. Whether it makes sense to define such data types is a different question.
- EXPRESS defines a couple of rules how a data type can be further specialized. This is important for re-declared attributes of entities.
- GENERIC data types can be used for procedures, functions and abstract entities.

Attribute (Figure 3.3-F-G):

An entity has attributes, describing the characteristics of this object. Attributes model the descriptive information units, and each attribute has a data type, which specifies the nature and values of the information unit. Data types can be the common computational types (Boolean, integer, real, string, enumeration), or entity types, or aggregates (set, list, array) of any of these. Also three different kinds of attributes exist. These are explicit, derived and inverse attributes.

- Explicit attributes are those which have direct values visible in a STEP-File.
- Derived attributes get their values from an expression. In most cases the expression refers to other attributes of THIS instance. The expression may also use EXPRESS functions.
- Inverse attributes do not add "information" to an entity, but only name and constrain an explicit attribute to an entity from the other end.

Lines (Figure 3.3):

Lines which are shown in the figure are used to connect the entity with its attributes. The names of the attributes are written above and along the lines.

Relationship (Figure 3.3):

Relationships are modeled as attributes whose data type is an entity type or an aggregate of an entity type. Some relationships are reified as entity types with role attributes whose values are the participating entities. A relationship expresses a dependency or interaction between two entities. A relationship has cardinality, which indicates the number of objects in each of the entities at either end of the relationship that may be involved in a particular instance of that relationship. A relationship also has a name. In the diagrammatic modeling notation in Figure 3.3, a relationship is shown as a solid or dashed thin line which is terminated by a circular arrowhead; the relationship name is written next to the line. The direction of a

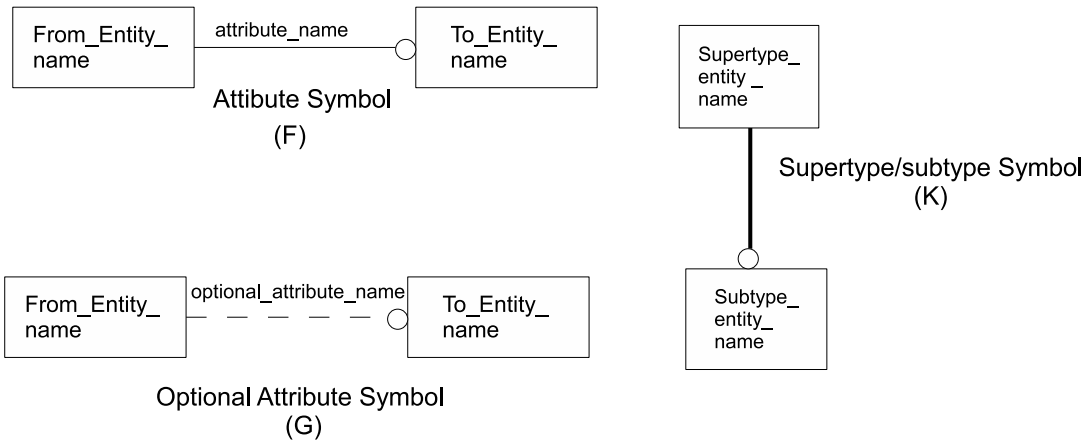


Figure 3.3: Express-G Line Symbols [Schenk and Wilson, 1994].

relationship is towards the arrowhead, which is important in that the name of the relationship must reflect its direction. A solid line indicates a compulsory relationship, whereas a dashed line indicates optional relationship.

Cardinality (Figure 3.4)

A cardinality string may be added to the relationship name, in which case it can take one of a number of forms. If a relationship has no cardinality string included in its name, then the cardinality is assumed to be exactly one. Cardinalities can be expressed in terms of sets, bags, lists and arrays. A set is an unordered variable length collection of unique items. A bag is an unordered variable length collection of not necessarily unique items. A list is an ordered variable length collection of not necessarily unique items; however there are possibilities to constraint the list to a list of unique items. An array is a fixed size collection of not necessarily unique items which can be accessed by an index. A cardinality can be shown as a string in the form C[m n], where C is one of S(set), B(bag), L(list) or A(array), where m is the lowest number of items allowed in the aggregation, and where n is the highest number of items allowed. If the cardinality is shown as m:n, then it is assumed to be as set (S). If the cardinality is shown as a single number (n), then it is assumed to be S[n n]. Note that an upper limit of "?" indicates 'unbounded'.

Inverse Relationship (Figure 3.4)

In most cases, an inverse relationship can be inferred directly from the original relationship. This relationship is indicated by writing (INV) in front of the name of the relationship (Figure 3.4).

Supertypes and Subtypes (Figure 3.3-K):

An entity can be defined to be a subtype of one or several other entities (multiple inheritance is allowed). A supertype can have any number of subtypes. It is very common practice in STEP to build very complex sub-supertype graphs (Figure 3.3-K). An entity instance can be constructed for either a single entity (if not abstract) or for a complex combination of entities in such a sub-supertype graph. For the big graphs the number of possible combinations is likely to grow in astronomic ranges. To restrict the possible combinations special supertype constraints got introduced such as ONEOF and TOTALOVER. Furthermore an entity can be declared to be abstract to enforce that no instance can be constructed of just this entity but only if it contains a non-abstract subtype. Algorithmic constraints.

Entity generalisation and specialisation: Two or more entities which have some (but not all) characteristics and/or behavior in common may be generalised into a supertype. Each of the entities that the supertype generalises is known as a subtype. Another interpretation of subtypes and supertypes is to consider that a supertype entity is specialised into a series of subtypes. Each of the subtypes is a specialisation which inherits all of the characteristics and behavior of the supertype, but adds new characteristics and/or behavior of its own. Note that it is possible for an entity to be both a supertype and a subtype simultaneously.

Rules: Entities and defined data types may be further constraint with `WHERE` rules. `WHERE` rules are also part of global rules. A `WHERE` rule is an expression, which must evaluate to `TRUE`, otherwise a population of an `EXPRESS` schema, is not valid. Like derived attributes these expressions may invoke `EXPRESS` functions, which may further invoke `EXPRESS` procedures. The functions and procedures allow formulating complex statements with local variables, parameters and constants - very similar to a programming language. Example shows that week value cannot exceed 7.

```
TYPE day_in_week_number = INTEGER;
WHERE
  WR1: (1 <= SELF) AND (SELF <= 7);
END_TYPE; -- day_in_week_number
```

3 EXPRESS-G

`EXPRESS-G` is a standard graphical notation for information models [Schenk and Wilson, 1994]. It is a useful companion to the `EXPRESS` language for displaying entity and type definitions, relationships and cardinality. This graphical notation supports a subset of the `EXPRESS` language. One of the advantages of using `EXPRESS-G` over `EXPRESS` is that the structure of a data model can be presented in a more understandable manner. A disadvantage of `EXPRESS-G` is that complex constraints cannot be formally specified. Figure 3.4 is an example.

Explanation: Figure 3.4 shows the visualisation of a concept Person in `EXPRESS-G` notation. Here, a person has several characteristics like a first name and a last name, an optional nickname, a special

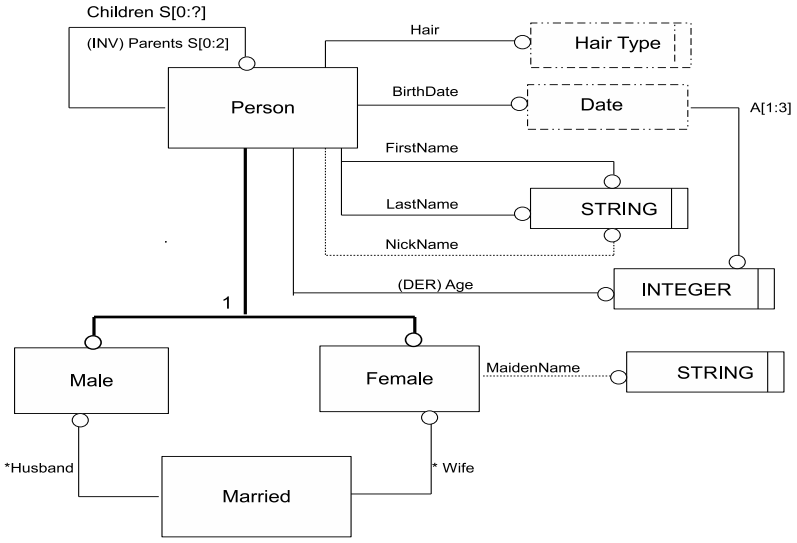


Figure 3.4: Concept Person in EXPRESS-G.

type of hair, a date of birth and implicitly a certain age. Age has been prefixed with (DER), for derived, to denote that it is a derived attribute. The enumeration HairType : bald, dyed, natural, wig has to be noted outside the diagram. In the example, a person is either female or male. If female, the person optionally has a maiden name. (This relation surely depends on the country's laws and can be regarded being sexistic.) A person may have children and up to two (living) parents, who naturally are persons, too. The attribute parents is defined as being inverse to children by a preceding (INV). In EXPRESS-G , an inverse attribute denotes a bi-directional relationship between two entities: an inverse attribute of an entity A references an entity B that itself references entity A [ProSTEP 1994].

A man and a woman may be married whereby in the chosen example polygamy as well as (for equality reasons) polyandry are forbidden through uniqueness constraints, i.e. that the values of husband and wife must be unique across all instances of entity Married. In EXPRESS-G , only the pure existence of these constraints can be displayed by prefixing Husband and Wife with an asterisk while the constraints (no_polyandry and no_polygamy) themselves have to be noted and defined outside the diagram, i.e. in EXPRESS .

4 Why is EXPRESS Used in Thesis Model

The aim of this thesis is to propose a model to represent user preferences. The purpose of this model is to be shared and generic. To make it sharable, it must be formally defined in order to remove ambiguities from its definition. We explain in this section why EXPRESS is well suited for this definition.

The EXPRESS modeling language is equipped with a powerful constraints language allowing to define precisely the semantics of the model. EXPRESS language is a data modeling language. It allows checking of properties, the correctness of defined limitations, establishing prototype, and provides opportunity to test the established prototype.

And by this means, for the Preference Model that was established for this research;

- required new types were established using "User Defined Type" definition which is a new advantage provided by EXPRESS language. Required properties were defined for the determined preference types, and relevant limitations were established,
- tests of the established prototype were substantiated,
- finally, providing global access to ontology model which was one of the aims of this research thesis was demonstrated on a model.

Preference model which was established by using the abilities of this language is explained in Chapter 5. Related EXPRESS code that was established for the model is represented in Appendix. Thus, Preference Model whose execution was demonstrated on OntoDB is shown as an authentic and working model in Chapter 6. Before presenting these chapters we introduce the chapter of OBDB.

Chapter 4

Ontology Based Databases: OBDB

Contents

Abstract. In the last decade, the notion of ontology based database (OBDB) has been developed in order to offer an infrastructure allowing management of both ontologies and their instances. This chapter describes in detail a model of architecture OBDB called OntoDB. Its implementation in an environment, consisting of the `EXPRESS` language and the PLIB ontology model is also briefly described.

1 Introduction

The Semantic Web is an effort by the W3C to enable integration of data sources across the Web. Ontologies have been defined to make the semantics of data explicit. In order to capture information semantics in a machine processable way, Web resources are described as ontology individuals. Such ontology individuals are called ontological data. As Semantic Web technologies become mature and standardized, they are applied to real-world applications. As a consequence, an increasing amount of ontological data is becoming available on the Web. To manage such data, Ontology Based DataBases (OBDBs) [Dehainsala et al., 2007b], [Dehainsala et al., 2007c], that store ontologies and their instance data in the same repository have been proposed.

OBDBs store both ontologies and ontology-based data in database schemas to get benefit of the functionalities offered by DBMSs (query performance, data storage, transaction management, etc.). Recently, several approaches and systems were proposed to store data and the ontologies describing their meanings (e.g. Sesame [Broekstra et al., 2002], RDFSuite [Alexaki et al., 2001], Jena [B.McBride, 2001], [Carroll et al., 2004], OntoDB [Dehainsala et al., 2007c], OntoMS [Park et al., 2007] and KAON [Bozsak et al., 2002]) in the same database. They have two characteristics. First, they allow manage both the ontologies and the data. On the other hand, they allow associating each data to the ontological concept, that defines its meaning.

In this chapter, Section 2 presents different Ontology Based Database (OBDB) approaches. Section 3 presents the OBDB model addressed in this thesis and introduces the OntoQL exploitation language. Section 4 explains why OntoDB is used in this thesis work. Section 5 concludes this chapter.

2 OBDBs Approaches

In the last decade, the notion of ontology based database (OBDB) has been developed [Dehainsala et al., 2007b], [Pierra et al., 2005] in order to offer an infrastructure, allows management of ontologies and their instances [Chong et al., 2005a], [Petrini and Risch, 2007]. At least these models store the ontology and its instances, but some of them also store the ontology model and extensively use meta modeling techniques. In this section different OBDB approaches are analyzed by presenting their architecture. They are classified into three main categories according to the number of schemas used.

2.1 Type I architecture

In OBDBs of Type 1, information is represented in a single schema composed of a unique triple table (subject, predicate, object) [Chong et al., 2005b], [Petrini and Risch, 2007]. This table, called *vertical table*, may be used for both ontology descriptions and instance data. For ontology descriptions, the three columns of this table represent respectively subject ontology element identifier, predicate and object ontology element identifier.

Figure 4.1 illustrates this approach. The triple1 (Student, subClassOf, Person) represents a subsumption relationship between classes Student and Person. For instance data, the three columns of this table represent respectively instance identifier, characteristic of an instance (i.e, property or class belonging)

and value of that characteristic. For example, the triple (Peter, grade, PhD) represents the fact that Peter has a PhD grade. Figure 4.1(a) presents a toy example of an ontology (upper part) with some instances (bottom part) as a graph. An extract of the corresponding vertical table is shown in Figure 4.1 (b).This approach raises serious performance issues, when queries require many self-joins over this table. The database structure is frozen. Since insertion/deletion operations of properties and instances are done easily. This representation is very simple but, it suffers from weak data typing and poor performance caused by several auto-join operations over the unique table. To optimize this architecture, clustering techniques need to be used [Agrawal et al., 2001]. This may dramatically cause maintenance overhead. Moreover, the ontology model being implicit, it needs to be hard encoded in the query language interpreter.

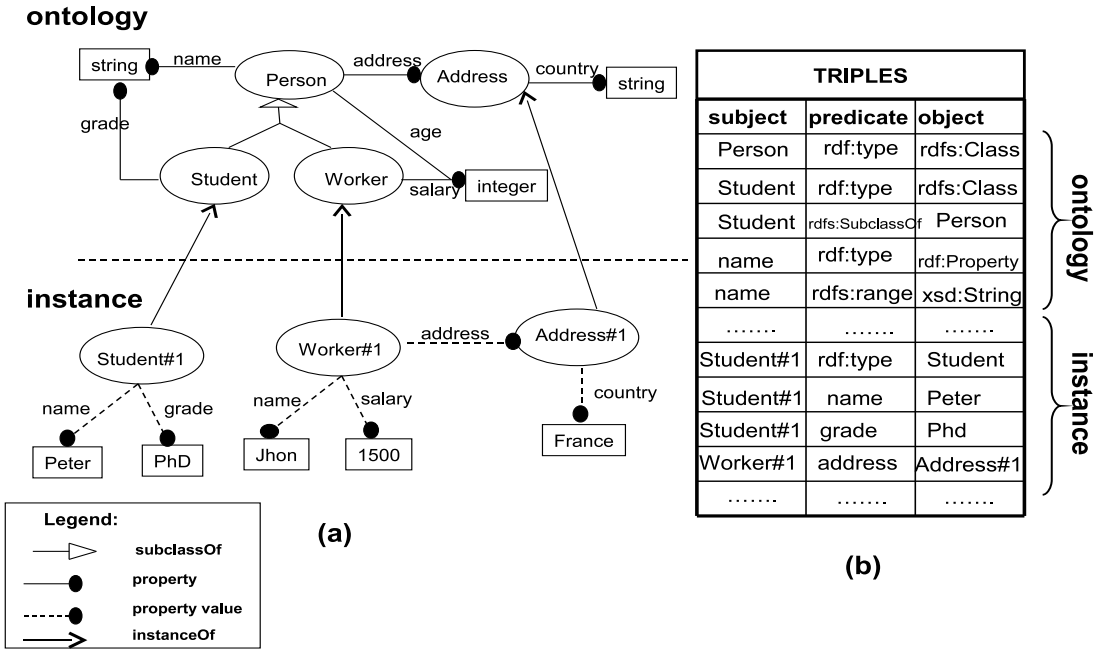


Figure 4.1: Type 1 OBDBs approach [Fankam et al., 2008].

2.2 Type II architecture

OBDBs of Type 2 store ontology descriptions and instance data in two different schemas [Alexaki et al., 2001], [Broekstra et al., 2002] separately. The schema for ontology descriptions depends upon the ontology model used to represent ontologies (e.g., RDFS, OWL, PLIB). It is composed of tables, that are used to store each ontology modeling primitive such as classes, properties and subsumption relationships. For instance data, different schemas have been proposed. A *vertical table* can be used to store instance data as triples. An alternative is to use a binary representation where each class is represented by an unary table and each property by a binary table. Recently, table per class representations (also called class-based representations) have been proposed. Table having a column for each property associated with value for at least one instance of a class is associated to each class.

Figure 4.2 presents an example of type 2 OBDBs that stores data of our previous example (see Figure 4.1). In this example, ontology descriptions are stored using a schema for RDFS ontologies.

It still has some drawbacks: (1) the ontology schema is based on the underlying ontology model and thus is static, and as a consequence (2) introduction of concepts originated from other ontology models is not allowed. In Sesame, for example, the structure of the ontology part is based on RDFS (tables include: class, property, domain, range, etc.), whereas different representations can be used for the data part: (1) A unique table of triples (like in type I architecture), which contains extensions of all concepts (classes and properties) of the local ontology. (2) A unary distinct table for each class of the ontology and a binary table for each property of the ontology. In this approach, the management of the ontology part and the data part is different. This architecture is more efficient. The second data representation scales quite well, especially, when queries refer to a small number of properties. Contrarywise, when each instance is described by a large number of properties, it does not scale.

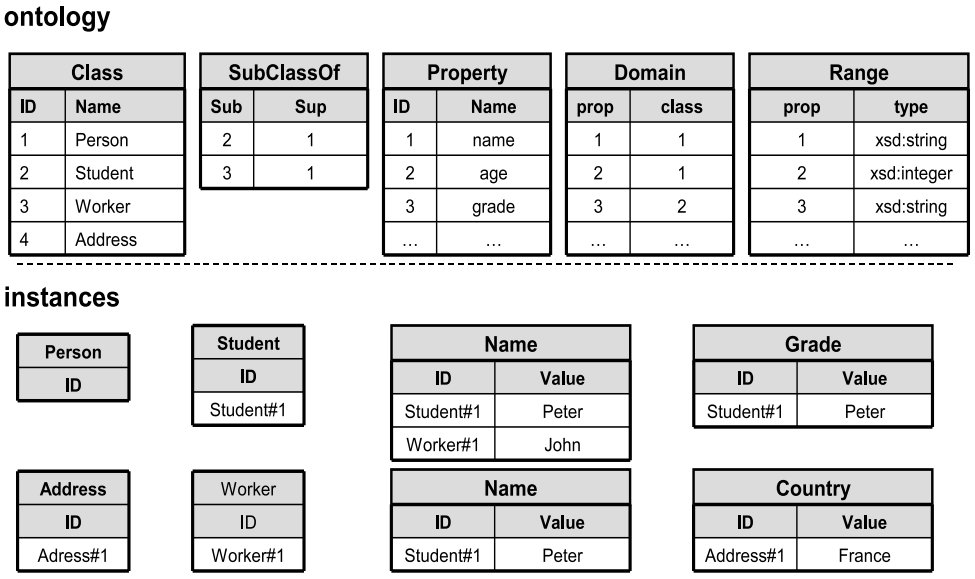


Figure 4.2: Type 2 OBDBs approach [Fankam et al., 2008].

2.3 Type III architecture

OBDBs of Type 3 architecture has been proposed for OntoDB [Dehainsala et al., 2007a, Pierra et al., 2005, Dehainsala, 2007], with PLIB as the underlying ontology model. An additional part, called the *meta schema* part, is introduced in Figure 4.3. Thus the database structure is defined by three schemas. The presence of the meta schema part offers flexibility of the ontology part, since it is represented as an instance of the meta schema. For the ontology schema, the meta-schema plays the same role as the one played by the system catalog in traditional databases. Indeed, meta-schema may allow: (1) generic access to the ontology, (2) support of evolution of the used ontology model, and (3) storage of different ontology models (OWL, DAML+OIL, PLIB, etc.). The possibility (2) is crucial for our work inorder to extend OBDB with preferences.

Meta-Schema

Entity		
ID	Name	SuperEntity
1	Resource	1
2	Class	1
3	Property	1

Attribute			
ID	Name	Domain	Range
1	name	1	1
2	domain	3	2
.....	.....	.....	.....

Type	
ID	Name
1	String
2	Entity#2

Figure 4.3: Type 3 OBDBs approach [Fankam et al., 2008].

3 OntoDB Ontology Based Database Model

In the 90s, to allow the exchange of electronic catalogues of industrial components, an ontology model for technical domain was developed and then published as an international standard known as PLIB [ISO13584, 1998], [Pierra, 2003a]. Then a model to exchange objects described in terms of such ontologies, was developed [Pierra, 2003a] and also standardized (ISO 13584). In the beginning of 2001, the PLIB model was finished and a new project called OntoDB [Dehainsala et al., 2007a, Pierra et al., 2005] was launched. It is aimed to store, exchange, integrate and process industrial catalogues modeled as ontology-based data, associated with a formal ontology. PLIB-based ontologies were first targeted. These ontologies are domain ontologies: they describe, by means of classes and properties, all the consensual entities of the target domain. Each property is defined in the context of a class, that constitutes its domain, and it has a meaning only for this class and its possible subclass(es). Then, the decision, to support also other ontology models like OWL or DAML+OIL has been taken.

OntoDB is implemented on top of the PostgreSQL 8.1 DBMS and the PLIB ontology model (POM), specified in the EXPRESS language, is used as underlying ontology model.

3.1 The PLIB ontology model

The PLIB ontology model [Pierra, 2003b] is designed to describe the entities existing in a field, through properties that characterize all entities of the domain. Some properties have a meaning only for a subset of objects in the area, classes are introduced, provided they are necessary to define the domain of certain properties. Each property is defined in a class entity, and only makes sense for this class and any subclasses. Class can be linked through the usual is-a subsumption relationship but also with a special relationship called caseof (is-a-case-of). This relationship allows a user to define its own ontology from an ontology shared and explicit correspondence (mapping [Bernstein et al., 00]) between these two ontologies. The PLIB ontology model is itself defined in the EXPRESS data modeling language [Schenck and Wilson, 1994].

A PLIB ontology has the following characteristics.

- *Conceptual*: each entry is unique and completely defined. The words that appear in its description clarify its meaning.
- *Multilingual*: each entry is associated with a code which is a universal identifier for identifying

the corresponding concept. Textual description aspects can appear in any number of languages.

– *Modular*: an ontology can referenced an another ontology for importing entities and properties without duplicating them.

– *Multi-representation*: once defined, a concept can be associated with an unlimited number of representation. A view that characterizes each representation is a concept represented in the ontology.

– *Consensual*: the conceptual model of ontologies PLIB has reached an international consensus and published under form of ISO and IEC standards. Ontologies conform to this model are developed either through a standard that requires an international consensus on the content, or by industrial consortia grouping a large number of partners.

3.2 OntoDB Architecture

The aim of OntoDB is to provide a scalable and evoluatative system to manage ontologies and their instances. OntoDB ensures models and their instance's persistency, whereas it associated language named OntoQL [Jean et al., 2005a], [Jean et al., 2006b],[Jean, 2007] allows to manage and query ontologies. Its architecture is composed of two main parts. The *ontology part* and the *content part*

The ontology part stores ontology definitions. It gathers the basic shared constructions of the PLIB [ISO13584, 1998], RDFS [Brickley and Guha, 2004] and OWL [Dean and Schreiber, 2004] ontology models. To implement the ontology part, the PLIB ontology model has been mapped to a logical schema by a program generator. It is based on defined transformation rules between EXPRESS concepts and SQL/DDL. The logical schema of the meta schema part is also generated automatically by re-using an object relational generator. Concretely, the generator receives as input parameter an EXPRESS meta model and returns a set of tables representing the meta model. Then the meta schema part is populated with the PLIB ontology model and with itself as data.

The content part stores the instances which descriptions and semantics are described by the stored ontologies. The data represents the objects in the area, who are described in terms of a class of belonging and a set of property values for this class.

Figure 4.4 shows the logical architecture of OntoDB. This architecture represents domain ontologies described in terms of classes and properties, and objects in the field, defined in terms of these ontologies. It is composed of four parts. Parts 1 and 2 are traditional parts available in all RDBMSs, namely the data part that contains instance data and meta-base part that contains the system catalog. Parts 3 (ontology) and 4 (meta-schema) specific our OntoDB.

- **Metabase Part (1).** This part, often called system catalog, is a part of traditional classical database. It consists of all system tables. These tables are those which the DBMS uses to manage and operate all data contained in the database. In OntoDB, it contains in particular the description of all tables and columns defined in the other three parts of this architecture.

- **Data Part (2).** The data part contains description of object instances (belonging to the ontology domain) described in terms of ontology class belonging and ontology property values. But, unlike

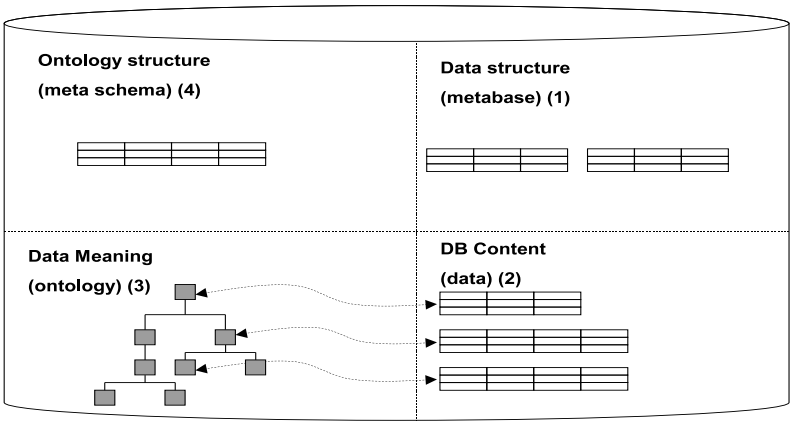


Figure 4.4: OntoDB Architecture.

individuals of description logic that may be described by any number of class belonging and by any existing properties (if they are not associated with specific constraints) thus making difficult storage indexing. In the OntoDB model instance, data must obey to two assumptions. (A1) Each instance must belong to one class, only called its base class (and to all of its superclasses). (A2) Each instance may be only described by properties that are applicable for its base class. With these two assumptions, each class may be associated with a table a view of which each row describes an instance that defines this class as its base class, and of which columns are the subset of applicable properties that were selected to constitute the schema of this class.

- **Ontology Part (3).** This part contains ontology definition as instances of the ontology model (that may be PLIB or any other model represented as a set of objets).

- **The Meta-Schema part (4).** The main objective of the meta-schema is to offer a programming interface allowing to access the current ontology model. This makes it generic according to ontology models. This part records the ontology model into a reflexive meta model. For the ontology part, the meta schema part plays the same role as the one played by the meta-base in traditional DBs. Indeed, this part may allow: (1) generic access to the ontology part, (2) support of evolution of the used ontology model, and (3) storage of different ontology models (OWL, DAML+OIL, PLIB, etc.).

To conclude this section, we can see that OntoDB's architecture has strong OBDB similarities with the architecture of metadata MOF (Meta Object Facility) [Kobryn, 99]. This architecture consists of four layers. Layer model M0 MOF architecture contains of the domain defined as instance of ontologies. Layer model M1 MOF architecture is OntoDB's conceptual model, subset of the ontology. The meta-model layer M2 corresponds to the ontology model, the layer meta-meta-model M3 (MOF model) is the meta-model of the ontology, itself reflexive. This architecture allows us integrate automatically [Bellatreche et al, 2003] [Bellatreche et al, 2004], to migrate and exchange bodies not necessarily defined in the ontology model. Moreover, it allows us to use the results of work performed under the MOF.

3.3 OntoQL Query Language for OntoDB

OntoQL [Jean, 2007], [Jean et al., 2005b], [Jean et al., 2006a] has been defined as an extension of SQL to exploit OBDBs. This language offers the possibility to query ontologies, contents (instances) and both ontology and content in parallel. Also it allows the modification of the meta-model level (i.e. the ontology model used to define ontologies) and ensures that such changes comply with the semantics of the system. This section gives specifications of this language.

3.3.1 The Data Def nition Language

The Data Definition Language (DDL) is used to create and destroy in an OBDB ontologies and the conceptual model of data subset of an ontologies. In OntoQL, the `CREATE, ALTER and DROP TYPE` where `TYPE` is a type of class, help to define the user classes and their properties. The syntax for creating a class and its properties is given by:

```
<class definition> ::= CREATE <entity id> <class id> [ <under clause> ]
[ <descriptor clause> ] [ <properties clause list> ]
<under clause> ::= UNDER <class id list>
<descriptor clause> ::= DESCRIPTOR ( <attribute value list> )
<attribute value> ::= <attribute id> = <value expression>
<properties clause> ::= <entity id> ( <property definition list> )
<property definition> ::= <prop id> <datatype> [<descriptor clause>]
```

Syntax explanation. The header of the instruction begins with the keyword `CREATE.` The element `<entity id>` specify the type of the class created. It is followed by the identifier of the created (`class <class id>`) and the possible list of its super-classes after the keyword `UNDER.` The body of this instruction is composed of several optional clauses. `DESCRIPTOR` clause can be used to describe the created class by specifying the attribute values (`<attribute value>`). Other clauses (`<properties clause>`) create, together properties (`<prop id>`) defined on this class. These clauses begin with <entity id> which supports the specification of the type of created properties.

The syntax for creating the extension of a class, i.e the conceptual model for its instances, is as follows:

```
<extension definition> ::= CREATE EXTENT OF <class id> (
<property id list> ) [<logical clause>]
<logical clause> ::= TABLE [<table and column name>]
<table and column name> ::= <table name> [( <column name list> )]
```

Syntax explanation. Several type tables can be associated with a user type, the direction of creating a table requires typed name the table created. This is not necessary for the instruction of creating a extension since it is unique for a given class. Accordingly, the header of this instruction can define an extension indicating only the class name (`OF` <class id>). The body of this instruction is composed of the element <property id list>. By default, the interpreter of such an instruction implements the extension created by

the logic level the *horizontal* representation, that is to say by a table (currently non-standard) including a column for each property of the extension.

The ALTER statement to modify existing classes has the following syntax:

```
<alter class statement> ::= ALTER <class id>
[ <descriptor clause> ] [ <alter class action> ]
<alter class action> ::= <add property definition> |
<drop property definition>
<add property definition> ::= ADD [<entity id>]
<property definition> [<descriptor clause>]
<drop property definition> ::= DROP <property id>
```

Syntax explanation. This instruction adds (ADD) or deletes (DROP) a property defined on a class. It allows users to change the description of a class in a DESCRIPTOR clause. The values of attributes specified in this clause override those that may have been previously defined. The semantics of this instruction is similar to editing a user type. Thus, deletion of an inherited property is not allowed. Similarly, it is not possible to delete a property if it is the only property defined on a class. The syntax to delete a class and the associated properties is as follows:

```
<drop class definition> ::= DROP <class id>
```

Syntax explanation. This statement removes the class identifier <class id> and proprieetes defined over the class. This class must not have sub-classes or extension and should not be used in a reference type.

3.3.2 The Data Manipulation Language

The Data Manipulation Language (DML) can be used to modify a user type throught the INSERT, DELETE and UPDATE clauses. The syntax of the INSERT clause is as follows:

```
<insert statement> ::= INSERT INTO <class id> <insert description and source>
<insert description and source> ::= <from subquery> | <from constructor>
<from subquery> ::= [ ( <property id list> ) ] <query expression>
<from constructor> ::= [ ( <property id list> ) ] <values clause>
<values clause> ::= VALUES ( <values expression list> )
```

Syntax explanation. This syntax allows to create one or more instances of a class. In SQL, as several table types can be associated with a user type, the INSERT statement requires to specify the name of the table type in which the bodies will be inserted. Since a class has only a single extension, only the identifier (<class id>) is required to determine the extension in which bodies should be inserted.

Modifying instances is same throught the UPDATE statement using the following the syntax:

```
<update statement> ::= UPDATE <class id polymorph> SET <set clause list>
[ WHERE <search condition> ]
<class id polymorph> ::= <class id> | ONLY (<class id>)
<set clause> ::= <property id> = <value expression>
```

Syntax explanation. Direct instances of a class (`ONLY <class id>`) or all instances of a class (`<class id>`) and values properties (`<set clause>`) can be modified.

The `DELETE` statement removes instances using the following the syntax:

```
<delete statement> ::= DELETE FROM <class id polymorph> [ WHERE <search condition> ]
```

Syntax explanation. As the `UPDATE`clause,`DELETE` clause focuses only direct instances of a class or its subclasses. Deleting an instance is possible if it is not referenced by a type reference.

3.3.3 The Ontology Def nition Language (ODL)

The ODL can create, modify and delete entities and attributes in the ontology model, that is considered by OntoQL. This language makes it possible to change the model of used ontologies.

```
<entity definition> ::= CREATE ENTITY <entity id> [ <under clause> ] <attribute clause>
<under clause> ::= UNDER <entity id list>
<attribute clause< ::= <attribute definition list>
<attribute definition> ::= <attribute id> <datatype> [ <derived clause> ]
<derived clause> ::= DERIVED BY <function name>
```

Syntax Explanation. This statement creates a new entity in the ontology model OntoQL as the creation of a user type. Thus, this new entity can be created as sub-entity of one or more other entities (`UNDER`). It is defined with a list of attributes, that can characterize its instances. By default, the values of these attributes are defined by a user. This instruction makes it possible to define attributes derived through the provision `<derived clause>`.The function of derivation of such attribute is indicated by the name of a user function (`<function name>`). It must be defined with language programming (SQL / PSM) associated to the DBMS on which the OBDB is located.

The modification of entities and attributes of the OntoQL ontologies can be achieved by the following syntax:

```
<alter entity statement> ::= ALTER ENTITY <entity id> <alter entity action>
<alter entity action> ::= <add attribute definition> | <drop attribute definition>
<add attribute definition> ::= ADD [ ATTRIBUTE ] <attribute definition>
<drop attribute definition> ::= DROP [ ATTRIBUTE ] <attribute id>
```

Syntax explanation. An entity can thus be modified by adding (`ADD`) or removing (`DROP`) attribute. The core ontology model is the basis on which the semantics of language OntoQL is based. It is not possible to delete an attribute of this model.

The removal of entities and attributes in the ontology model is achieved by the following syntax :

```
<drop entity statement> ::= DROP ENTITY <entity id>
```

Syntax explanation. This statement removes the entity <entity id> and all its attributes. An entity can not be deleted if it is not the core model and if it is not referenced by an other entity. This is the case, if it has sub-entities or whether it is used to define a co-domain attribute. It can not be deleted if that entity is a sub-entity and a Class # their bodies are associated with an extension.

3.3.4 The Ontology Manipulation Language (OML)

The OML should allow to create, edit and delete elements of an ontology such as its classes and properties. Having already a syntax defined for creating classes and properties of ontology (DDL for data-based ontological), this must be taken into account while analyzing the remaining needs. Use a OML to create a class and a property would require several instructions INSERT.Note that it would in a normal DBMS, create a table by performing insertions tables in the metabase.

The OML can create elements in an ontology using the following syntax:

```
<insert statement> ::= INSERT INTO <entity id> >insert description and source>
<insert description and source> ::= <from subquery> | <from constructor>
<from subquery> ::= [ ( <attribute id list> ) ] <query expression>
<from constructor> ::= [ ( <attribute id list> ) ] <values clause>
<values clause> ::= VALUES ( <values expression list> )
```

Syntax explanation. This can add instances to the entity <entity id> . These instances can be defined by specifying the set of values of attributes (<from constructor>). They can also be the result of a request OntoQL (<from subquery>). The entities and attributes used are either those model ontologies summarized in Table 4.1 for the main entities.

Entity	Attributes
#Ontology	#oid, #namespace
#Concept	#oid, #code, #name, #definition, #definedBy
#Class	#oid, #code, #name, #definition, #directSuperclasses, #definedBy
#Property	#oid, #code, #name, #definition, #scope, #range, #definedBy
#Datatype	#oid
#RefType	#oid, #onClass
#PrimitiveType	#oid
#CollectionType	#oid, #ofDatatype, #maxCardinality

Table 4.1: The main entities of the ONTOQL

The items created in an ontology can be modified by using the following syntax:

```
<update statement> ::= UPDATE <entity id polymorph> SET <set clause list>
```

```
[ WHERE <search condition> ]
<entity id polymorph> ::= <entity id> | ONLY (<entity id>)
<set clause> ::= <attribute id> = <value expression>
```

Syntax explanation. This syntax allows to update the direct instances (`ONLY`) or also indirect by allows without the word Key `ONLY` assigning new attribute values to an entity.

The elements of ontologies can be deleted using the following syntax:

```
<delete statement> ::= DELETE FROM <entity id polymorph> WHERE <search condition>
```

Syntax explanation. This statement eliminates entity also directly or indirectly with respect to a given predicate (`<search conditioning>`).

4 Why OntoDB Is Used in Thesis Model

In order to validate our thesis proposition we needed to have an infrastructure allowing to encode ontologies with a manipulation language. The OntoDB ontology based database and the OntoQL language have been chosen for this purpose. OntoDB ensures models and their instances' persistency, whereas OntoQL allows to manage and query ontologies. We chose the OBDB OntoDB for two main reasons.

1- The first reason is that, as previously observed, other existing OBDBs only deal with a single ontology model (RDFS, OWL, PLIB,...), or, with possible semantic-compatible ontology models (RDFS/OWL). Therefore, none of them is a good candidate for fields, where applications require constructs from ontology-models with different semantics. OntoDB provides a more adequate environment, i.e, based on an ontology model integrating constructs from several ontology models like RDFS, OWL and PLIB. Based on the classification Section 2 type III architecture is more flexible. Thus this architecture seems a good candidate for our proposed approach.

2- The second reason is that the currently defined OBDB does not deal with the representation of the non-functional aspects related to the ontology models. By non-functional aspects, we mean concepts that are capable to describe externally defined properties like quality, preferences or security are meant. Indeed, most of the well known ontology models like OWL, PLIB, etc. do not provide with such resources to represent such concepts. Each time, non-functional concepts are introduced, ad hoc concepts or extensions are introduced in the ontology models like OWL, PLIB, etc. Specific attributes, like note or remark or a particular property, are used to encode these non-functional aspects. Rather than extending a specific ontology model, our proposal consists in introducing a side model to describe the non-functional concepts related to preferences together with the ontology model inside an OBDB.

5 Conclusion

In this chapter we first presented the ontology based database that support the description of an ontology together with its instances. Then we particularly focus on the OntoDB ontology based database. OntoDB

has there main functionalities:

(1) a storage of a domain ontology and database content in the same repository, (2) the possibility of querying databases at ontology level, and (3) an automatic integration of hetero- geneous data sources referencing/extending the same domain ontology.

Finally, were explained the usage of the ontology based database OntoDB. OntoDB architecture is an OBDB Type 3 architecture. Thus it provides a meta-schema structure. Before presenting the idea of the extension of meta-schema structure, Preference Model is proposed in Chapter 5. This model which is an external model, is created independently of the ontology model.

Part III

OUR PROPOSAL: HANDLING PREFERENCES AT THE ONTOLOGY LEVEL

Chapter 5

Proposed Model of Preferences

Contents

Abstract. Providing personalized access to information is fundamental, especially in the context of the Web where a huge amount of data is available. Many approaches have been tried to represent and exploit user preferences. Usually these approaches focus on a particular application leading to difficulties to share and reuse the captured preferences. In this chapter we propose a generic model based on various models proposed in the Database and Semantic Web communities. The idea consists in raising preference handling at the ontology model level instead of at the logical model level.

1 Introduction

Current Web information systems have to manage a huge amount of data. This is particularly the case in domains such as the Semantic Web or engineering sciences where numerous digital data have been defined. Indeed, because standard ontology models exist (OWL , RDF Schema [Brickley and Guha, 2004], PLIB [Pierra, 2003a]) more and more ontologies have been designed and as a consequence, the amount of data described by ontologies (ontological data) has increased. Usually Semantic Web information systems return numerous results in response to user requests that must be sorted and filtered in order to find the relevant ones. This problem is fundamental for many applications especially in the e-commerce domain. As a solution to this problem, many approaches have proposed to capture and exploit user preferences in order to adapt to the user results produced by query processing.

Preferences represent the basic notion for any decision support activity. One of the principal tasks within a decision aiding process is to model preferences in such a way that it is possible to derive a final recommendation for the decision maker. Moreover, they are complex to evaluate and according to the user goals and a current task, they should be evaluated in the context where they are expressed. Preference modeling and preference-based search is a popular approach for helping consumers to find their desired items. The user preference modeling plays an important role in current web applications. Users make decisions by considering a set of criteria that involve attributes. For example, a criterion could be as tiring as a trip is likely to be, or how well its schedule fits the tasks to be accomplished. Each criterion is a function of one or several attributes. Many criteria are simple functions of a single attribute: for example, whether the arrival time is early enough for a 18:00 meeting, or whether the airline fits the user's preference. Others can be more complex: for example, how tiring a trip is depends on the total travel time, the departure and arrival times, the number of stops, etc.

In decision-making and decision-support tasks, a model of the user' s preferences is required to make good decisions or to suggest good alternatives. Representations for preference models have been studied extensively in the literature on multi-attribute utility theory (e.g. Keeney and Raiffa, 1976), which provides compact representations and elicitation techniques for preference models, but generally assumes that the model is built by a human expert. Problem solvers like AI planning algorithms generally assume that the complete preference model is provided as an input, but this is not a good approach to interactive problem solving in complex domains. Ahead-of-time elicitation demands a tremendous amount of information from the user, most of which will be irrelevant to solving the particular problem at hand. An alternative approach has been to infer a user model automatically over multiple interactions with the user that is used to support decision making and information filtering (e.g. [Gerhard et al., 1997] and [Mostafa et al., 1996]). The ontology modeling approaches solve the important part of capturing the requestor's preference by formally specifying the considered selection criteria with semantic vocabulary and a classification structure[Chaari et al., 2008].

However, in most existing information systems, preferences are not modeled explicitly. They are often hard coded and disseminated through applications that exploit these information systems. As a consequence, user preferences can not be shared between web information systems. Users preferences have to be defined and updated for each application, which is a burden for users and yields to another layer of heterogeneous modeling.

In this chapter we will describe our proposal of a model for representing preferences in a declarative,

domain independent and machine interpretable way. In order to represent user's preferences, we propose a modular, sharable and generic model to:

- make it sharable. We have defined it formally using the EXPRESS modeling language in order to remove ambiguities from its definition. Indeed,the EXPRESS modeling language is equipped with a powerful constraints language allowing to define the semantics of the model precisely;
- show it is generic. We have studied different approaches that have been proposed in the Database as well as in the Semantic Web communities and we have generalized them to define the proposed model;
- present modularity.We have chosen to define the preference model separated from the data and instances of the exploited information system.

Next section defines resources we need to defined our Preference Model. They are named Preference Model Resource Definition. Section 3 presents our Preference Model, and Section 4 explains the link between preference and ontological data. Section 5 concludes this chapter by summarizing the main results.

2 Resource Def nition

The term *resource* was first introduced to refer to targets of "Uniform Resource Locators" (URLs), but its definition has been further extended to include the reference to any Resource Identifier (RFC 3987 & RFC 3986). It is used often, specifically in relation with the World Wide Web and the W3C's semantic web activity (in standards such as Resource Description Framework (RDF), Uniform Resource Identifier (URI) and others). Nowadays, the *resource* term represents any identified object on the Web: a Web site, a Web page, a part of a Web page or Ontology concepts identified by URI. In this section, firstly we will give generic ontological resource definition, then we will explain our resource definition.

2.1 Resource Def nition in Ontology Def nition Meta-Model(ODM)

Ontologies are formal organization of domain knowledge to formally describe the semantic concepts in terms of classes and properties [Gruber and Olsen, 1994]. One of the most important components of ontologies is concept hierarchy. It models the information on the domain of interest in terms of concepts and subsumption relationships between them. The Ontology Definition Meta-Model (ODM) has been used as a basis for ontology development [Djuric et al., 2003],[Dragan Gasevic, 2006]. The corresponding ODM concepts are modeled by (MOF) [OMG, 2002]. MOF is a self-defined language intended for defining meta models. In term of Model Driven Architecture (MDA) a meta model makes statements about what can in the valid models of a certain modeling language be expressed.

MDA provides a solid basis for defining meta models of any modeling language, so it is the straight choice to define an ontology-modeling language in MOF. In fact, a meta model is a model of a modeling language. The MDA's meta model layer is shown on Figure 5.1. It is usually marked as M3, M2, M1, M0.

- M3: the MOF,

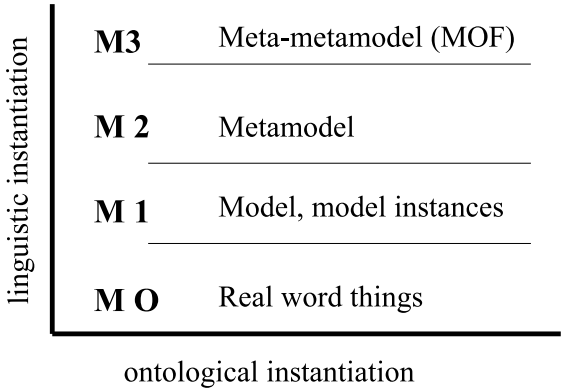


Figure 5.1: The four-layer Model Driven Architecture [Dragan Gasevic, 2006].

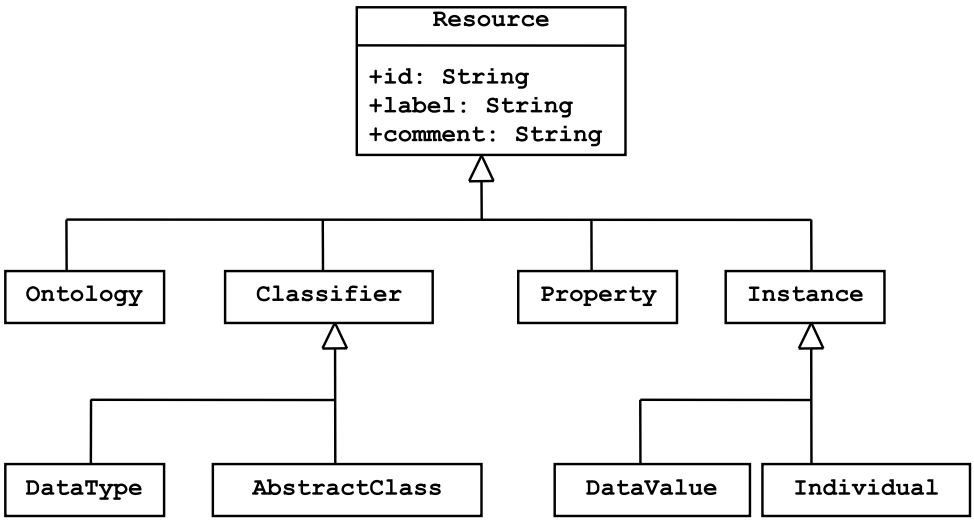


Figure 5.2: The Hierarchy of Basic Ontology Concepts [Brickley and Guha, 2004].

– M2: a MOF class model, specifying the classes and associations of the system being modeled.

– M1: an instance of a M2 model, describing a particular instance of the system being modeled.

– M0: ground individuals. A population of instances of the classes.

A detailed description of MOF can be found in OMG's MOF specification document [OMG, 2002], [Dragan Gasevic, 2006]. RDF, RDFS and their concepts are described in detail in W3C documents [Brickley and Guha, 2004].

Figure 5.2 briefly overviews the basic ODM concepts. In this figure the corresponding ODM concepts are modeled by MOF.

RESOURCE. Resource is one of the basic RDF concepts. It represents all things described by RDFS. Compared to ontology concepts, it can be viewed as a root concept, the Thing. In RDFS, Resource is defined as an instance of MOF Class. It is the root class of most other basic ODM concepts that will be described: Ontology, Classifier, Property, Instance, etc.

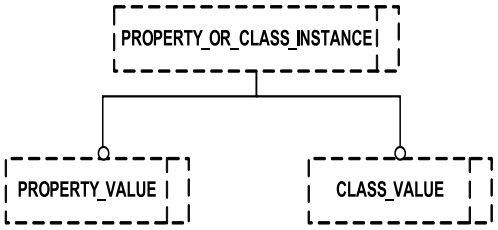
ONTOLOGY. Ontology is a concept that aggregates other concepts (Classes, Properties, etc.). It groups instances of other concepts that represent similar or related knowledge.

CLASSIFIER. Classifier describes some general concept that has its `Instances` (Individuals and DataValues).On the other hand, a Property describes some generic characteristics that can describe that Classifier and possibly other Classifiers.

PROPERTY. Ontology Class attributes or associations are represented through properties. A property is a relation between a subject resource and an object resource. Therefore, it might look similar to a concept of attribute and association in traditional, object oriented sense. In ODM, Property is an instance of MOF Class that inherits from Resource .

2.2 Resource Def nition in Preference Model

The first part of our model definition consists of resource definitions usefull to define our Preference Model. The first definition called `PROPERTY_OR_CLASS` resource, is introduced in order to attach a preference to an ontology. The seconed definition `PROPERTY_OR_CLASS_INSTANCE` resource is used to define specific instances of the ontology.



PROPERTY_OR_CLASS_INSTANCE

```
TYPE PROPERTY_OR_CLASS_INSTANCE
   SELECT (PROPERTY_VALUE, CLASS_VALUE);
END_TYPE;
```

Figure 5.3: Ontology Resource Definition

Model. In Figure 5.3, our resource definition is described as an `EXPRESS` entity. As it is shown, it is a select type (union of types) between the `CLASS_VALUE` type and `PROPERTY_VALUE` type. These types are in any existing ontology model. Thus, preferences are expressed on generic property or class instances. `CLASS_VALUE` is used to define a set of individuals and these individuals are defined by a class

identifier (an URI reference). Also PROPERTY_VALUE is used to describe some generic characteristic with their value and identifier.

Example. Let's suppose a Tourism Ontology with a class Hotel and a property starRate.

- *HotelMercury*, an instance of an hotel is an example of CLASS_VALUE.
  #1=CLASS_VALUE ('HotelMercury')
- *starRate=5*, property value of an hotel is an example of PROPERTY_VALUE.
  #2=PROPERTY_VALUE ('starRate', 5)

3 Preference Model for User Preferences

With the previous defined resources, we are now able to define our proposed Preference Model. This model collects different preference types from literature. In this section we propose our preference modeling approach by defining and illustrating each of its components using EXPRESS-G.

An overview of our approach is shown in Figure 5.4. The root entity of the Preference Model is PREFERENCE. And each PREFERENCE entity is associated with a PREFERENCE_URI.

The left part of the figure shows CONTEXT_PREFERENCE_DEFINITION. Context is a general term used to capture any information that can be used to characterize the situation of an entity. In this figure context is modeled as a set of multidimensional attributes. The formal definition of this model is given in Section 3.4.

The right part of the figure illustrates how PREFERENCE_DEFINITION can be resolved according to an interpretation. An interpretation is an explanation of the meaning of some object of interest. In Preference Model, INTERPRETED_PREFERENCEs are those preferences that can be given an interpretation by means of an evaluation. The nature of their definition depends on the attached interpretation function. Also UNINTERPRETED_PREFERENCES are used to define as an enumeration of a set of properties and classes values that are picked from an ontology without any constraint on the chosen values.

The following subsection details the Preference Model in order to express preferences on any set of data semantically described by an ontology.

3.1 Preference URI

In our model each preference is associated with an URI defined by PREFERENCE_URI entity. Its specification in EXPRESS (PREFERENCE_URI) is shown in Table 5.1.

Model. The PREFERENCE_URI entity characterizes a preference with a set of attributes. These attributes are *code*, *name* and *classification*:

- *the attribute code* gives a unique code e.g. any http address) for PREFERENCE_URI;
- *the attribute name* is a linguistic term in nature describing the PREFERENCE_URI;
- *the attribute classification* associates a category (e.g. cost, star rating, or distance) to a preference.

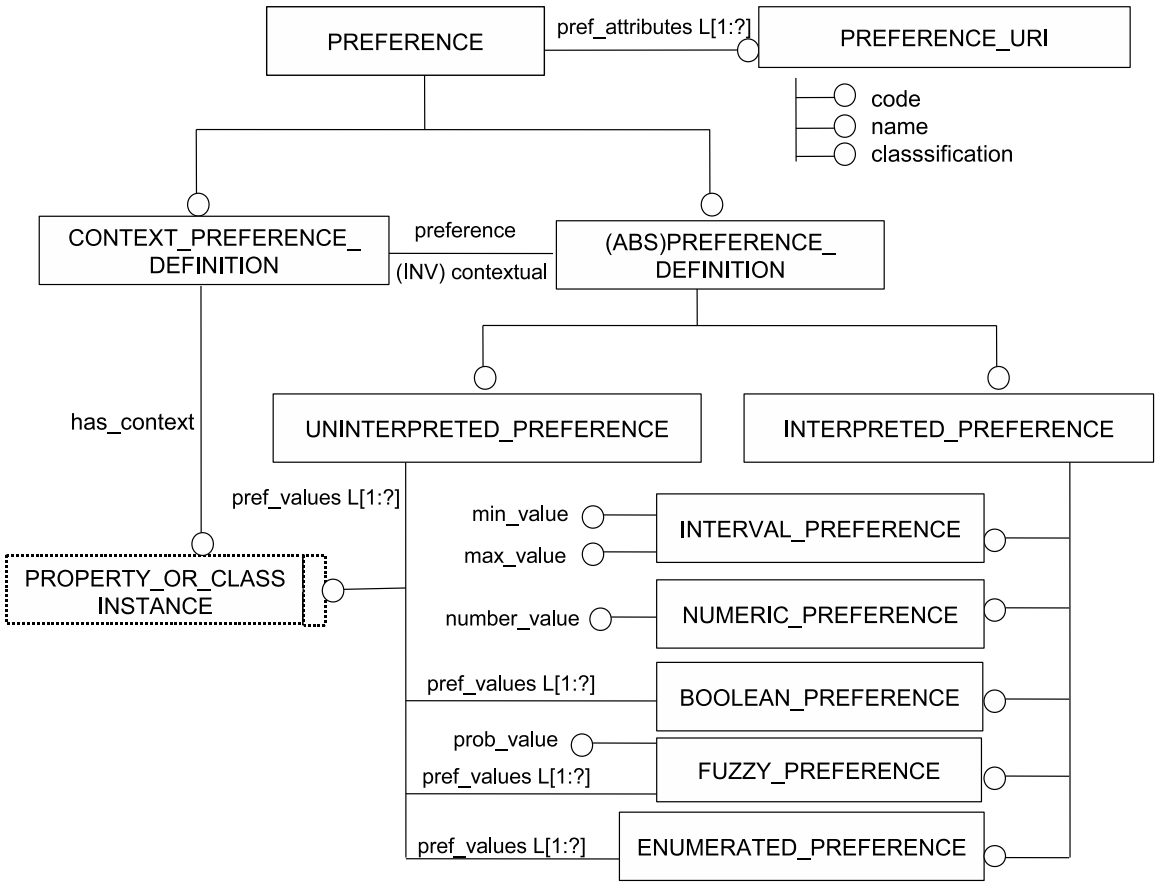


Figure 5.4: Preference Model Representation in EXPRESS-G

Table 5.1: Preference_URI.

```
ENTITY PREFERENCE_URI
 code:  INTEGER;
 name:  STRING;
 classification STRING;
END_ENTITY;
```

```
#2= PREFERENCE_URI(100,'cheap','cost');
#3= PREFERENCE_URI(101,'expensive','cost');
#7= PREFERENCE_URI(84,'low level','starRating');
#8= PREFERENCE_URI(85,'high level','starRating');
```

The PREFERENCE_URI entity is separately defined and could be introduced by a more general knowledge model in order to be reused. An ontology for example can be used to give a more precise description of the semantic annotation of a preference. Indeed, the classification attribute could be a reference to an ontology describing the different preference classifications.

Example. Lower part of Table 5.1 presents samples of PREFERENCE_URI.

- *#2 and #3* are two PREFERENCE_URI with code 100 and 101 means "cheap" and "expensive" and "cost" classification.
- *#7 and #8* are two PREFERENCE_URI with code 84 and 85 means "low level" and "high level" and "starRating" classification.

3.2 Interpreted Preferences

Preferences may be either interpretable or non-interpretable. Interpreted preferences are those preferences that can be associated to an evaluation or interpretation procedure. For example, we can interpret the preference *cheap(x)* as being *price(x)<=20*. The idea is to define preferences that are associated to data types that are valuable and to whom there exists an order relation. Before we can model interpreted preferences we have to consider the various possibilities. We have identified five types of preferences. Let us go through the identified interpreted preferences.

3.2.1 Enumerated Preference

Enumeration allows to declare a class by extension, it shows the population instances in a set. In mathematics and theoretical computer science, the broadest and most abstract definition of an enumeration of a set is an exact listing of all of its elements (perhaps with repetition). Table 5.2 gives the definition of ENUMERATED_PREFERENCE with examples.

Model. In our model ENUMERATED_PREFERENCE type corresponds to the enumeration of individuals taken in an ontology that interprets a given preference.

Table 5.2: Enumerated_Preference.

```
ENTITY ENUMERATED_PREFERENCE
   SUBTYPE OF (INTERPRETED_PREFERENCE);
   pref_values: LIST [1:?] PROPERTY_OR_CLASS_INSTANCE;
   pref_attributes: LIST[1:?] OF PREFERENCE_URI;
END_ENTITY;
```

```
#40= ENUMERATED_PREFERENCE([CLASS_VALUE('HotelFormule1'),
CLASS_VALUE('HotelPremiere')], [(#2), (#7)]);
#41= ENUMERATED_PREFERENCE([CLASS_VALUE('HotelHilton'),
CLASS_VALUE('HotelMercury')], [(#3), (#8)]);
```

– *The attribute pref_values* takes value from PROPERTY_OR_CLASS_INSTANCE resource defined in Section 2.2.

– *The attribute pref_attributes* associates an ENUMERATED_PREFERENCE with list of Preference_URI identifier.

Example. In Table 5.2, *HotelFormule1*, *HotelPremiere*, *HotelHilton* and *HotelMercury* are shown as domain ontology instances. First two hotels (both of them are 2 stars and they have only airCond and tv.) and last two hotels (both of them are 5 stars and they have golf and casino) have similar characteristics.

- *#2* and *#7* are two PREFERENCE_URI. They are used to describe *cheap* and *low level* preferences. In this example we could define *cheap and low level hotels* as being *HotelFormule1,HotelPremiere*. *#40* is given as example to ENUMERATED_PREFERENCE
- *#3* and *#8* are other two PREFERENCE_URI. They are used to describe *expensive* and *high level* preferences. In this example we could define *expensive and high level hotels* as being *HotelHilton, HotelMercury*. *#41* is given as example of ENUMERATED_PREFERENCE.

3.2.2 Numeric Preference

Numeric type can be discrete, meaning that the possible values are constrained to be one of a fixed set of values.

Model. In this model NUMERIC_PREFERENCE are interpreted by numeric values. This type of preference is specified in Table 5.3.

– *number_value* is a defined type based on integer.

– *The attribute interpreted_by* associates a NUMERIC_PREFERENCE with a set of *number_value*.

– *The attribute pref_attributes* associates a NUMERIC_PREFERENCE with a list of PREFERENCE_URI identifier.

Example. In this example we assume that the rating of a hotel can be *1, 2, 3, 4 5 or 6 stars* in a given tourism domain. *Interpreted_by* attribute takes value from this rating list and PREFERENCE_URI attribute describes the definition of preference.

Table 5.3: Numeric_Preference.

```
TYPE NUMBER_VALUE= NUMBER;
   END_TYPE;
```

```
ENTITY NUMERIC_PREFERENCE
   SUBTYPE OF (INTERPRETED_PREFERENCE);
   interpreted_by:  LIST[1:?]  OF NUMBER_VALUE;
   pref_attributes:  LIST[1:?]  OF PREFERENCE_URI;
 END_ENTITY;
```

```
#27= NUMERIC_PREFERENCE([1,2], [(#7)]);
#29= NUMERIC_PREFERENCE([5,6], [(#8)]);
```

- *#7* is PREFERENCE_URI. It is used to describe *low level* preference. In this example we could define *low level* hotel as being *1 star and 2 star hotels*. *#27* is given as example to NUMERIC_PREFERENCE.
- *#8* is also PREFERENCE_URI. It is used to describe *high level* preference. In this example we could define *high level hotels* as being *5 star and 6 star*. *#29* is given as example to NUMERIC_PREFERENCE.

3.2.3 Interval Preference

In mathematics, interval is a set of real numbers with the property that any number that lies between two numbers in the set is also included in the set. They are meaningful in any (totally or partially) ordered set.

Table 5.4: Interval_Preference.

```
ENTITY INTERVAL_VALUE
   min_value:  REAL;
   max_value:  REAL;
   WHERE min_value<max_value;
 END_ENTITY;
```

```
ENTITY INTERVAL_PREFERENCE
   SUBTYPE OF (INTERPRETED_PREFERENCE);
   interpreted_by:  INTERVAL_VALUE;
   pref_attributes:  LIST[1:?]  OF PREFERENCE_URI;
END_ENTITY;
```

```
#12= INTERVAL_VALUE(45, 60);
#17= INTERVAL_VALUE(90, 100);
#100= INTERVAL_PREFERENCE((#12), [(#2)]);
#110= INTERVAL_PREFERENCE((#17), [(#3)]);
```

Model. In our model INTERVAL_PREFERENCE are used to interpret a low and up values of preferences. Its definition is modeled on Table 5.4.

– *The type INTERVAL_VALUE* represents an interval using the two integer attributes.

– *The attribute interpreted_by* associates an `INTERVAL_PREFERENCE` with a `INTERVAL_VALUE.`

– *The attribute pref_attributes* associate with list of `PREFERENCE_URI` identifier.

Example. In this example, the *cheap* and *expensive* preferences can be respectively defined by the [45-60] and [90-100] intervals for "cost" classification of `PREFERENCE_URI.`

- *#2* is a `PREFERENCE_URI.` It is used to describe *cheap* preference. *Cheap hotel* price is defined as *[45-60]*. *#100* is given as example to `INTERVAL_PREFERENCE.`
- *#3* is other `PREFERENCE_URI.` It is used to describe *expensive* preference. *Expensive hotel* price is defined as *[90-100]*. *#110* is given as example to `INTERVAL_PREFERENCE.`

3.2.4 Fuzzy Preference

Sometimes preferences can be defined using probabilistic values. So we integrate fuzzy logic approaches in our model. `Fuzzy_Preferences` associate probability values to a given preference.

Table 5.5: Fuzzy_Preference.

```
TYPE PROB_VALUE= REAL;
   WHERE ((SELF>0) AND (SELF<1));
END_TYPE;
```

```
ENTITY FUZZY_PREFERENCE
   SUBTYPE OF (INTERPRETED_PREFERENCE);
   interpreted_by:  PROB_VALUE;
   pref_values:  LIST [1:?]  PROPERTY_VALUE;
   pref_attributes:  LIST[1:?]  OF PREFERENCE_URI;
 END_ENTITY;
```

```
#91=FUZZY_PREFERENCE((PROB_VALUE(0.10),
[PROPERTY_VALUE('starRate', 1)],[(#7)]);
#92=FUZZY_PREFERENCE((PROB_VALUE(0.95),
([PROPERTY_VALUE('starRate', 5),
PROPERTY_VALUE('golf', 'True')]), [(#3),(#8)]));
```

Model. The extension of boolean space by fuzzy logic makes possible the use of the weights assigned to the terms to evaluate their arguments. The result is no longer boolean, but a value between 0 and 1 corresponding to the estimated degree to which the given logical expression matches the given domain. Its `EXPRESS` definition is given on Table 5.5.

– *The defined type PROB_VALUE* has float value between 0 and 1.

– *The attribute interpreted_by* associates a `FUZZY_PREFERENCE` with a `PROB_VALUE`.

– *The attribute pref_values* takes value from `PROPERTY_OR_CLASS_INSTANCE resource defined in Section 2.2.`

- *The attribute pref_attributes* associates a FUZZY_PREFERENCE with list of PREFERENCE_URI identifier.

Example. In this example,

- *#7* is a PREFERENCE_URI. It is used to describe *low level* preference. *Low level hotel* is defined with *starRating* property value as 1. This example associates with 0.10 probability value in order to complete user's weak preference.
- *#3* is a PREFERENCE_URI which is used to describe *expensive* and #8 is used to show *high level* preferences. *High level hotel* is defined with *starRate* property value as *5* and *Expensive hotel* is defined with *golf* property value as TRUE. Thus, *#92* is given as example to FUZZY_PREFERENCE with 0.95 probability valueto indicate user's strongest preference is *high level and expensive hotel*.

3.2.5 Boolean Preference

BOOLEAN_PREFERENCE is discrete, with possible values of 1 and 0. The aliases 'TRUE' and 'Yes' are accepted for the value 1, and 'False' and 'No'for 0.

Table 5.6: Boolean_Preference.

```
ENTITY BOOLEAN_PREFERENCE
  SUBTYPE OF (INTERPRETED_PREFERENCE);
  interpreted_by: LIST[1:?] OF PROPERTY_VALUE;
  pref_attributes: LIST[1:?] OF PREFERENCE_URI;
END_ENTITY;
```

```
#21= BOOLEAN_PREFERENCE([PROPERTY_VALUE('tv', 'TRUE'),
PROPERTY_VALUE('airConditioner', 'TRUE')]),[(#2)]);
#22= BOOLEAN_PREFERENCE([PROPERTY_VALUE('casino', 'TRUE'),
PROPERTY_VALUE('golf', 'TRUE')]), [(#3)];
```

Model. BOOLEAN_PREFERENCE is specified on Table 5.6. In the model this preference type expresses a list of property values that a user prefers. It corresponds to the availability of this property.

- *The attribute interpreted_by* takes values from PROPERTY_VALUE. It is defined in Section 2.2.
- *The attribute pref_attributes* associates a BOOLEAN_PREFERENCE with a list of PREFERENCE_URI identifier.

Example. The proposed example in Table 5.6 defines a BOOLEAN_PREFERENCE on hotel room. This preference is defined as the presence of same values for the property *accomodation* (with or without *TV, airConditioner, golf, jakuzi*, etc.).

- *#2* is used to describe *cheap* preference. *Cheap hotel* is defined with only *t*v and *airConditioner* property values. *#21* is given as example of BOOLEAN_PREFERENCE .

- *#3* is used to describe *expensive* preference. *Expensive hotel* is defined with *golf* and *casino* property values. *#22* is given as example of BOOLEAN_PREFERENCE .

3.3 UnInterpreted Preferences

By uninterpreted preferences, we mean those preferences that are enumerated by a given user or a system designer without any associated interpretation procedure.

UNINTEPRETED_PREFRENCE type corresponds to a set of property or class values (instances) of an ontology that are selected as being preferred. There is no rationale for choosing these instances. They may be chosen, for different purposes The EXPRESS resources describing such a preference are described on Table 5.7.

Table 5.7: UnInterpreted_Preference.

```
ENTITY UNINTERPRETED_PREFERENCE
   interpreted_by:  LIST[1:?]  OF PROPERTY_OR_CLASS_INSTANCE;
   pref_attributes:  LIST[1:?]  OF PREFERENCE_URI;
END_ENTITY;
```

```
#47= UNINTERPRETED_PREFERENCE (([CLASS_VALUE('HotelIBIS'),
PROPERTY_VALUE('pizza_margarita', 'TRUE')]), [(#2)]);
```

Model. Table 5.7 defines attributes of the UNINTERPRETED_PREFERENCE entity.

- *The attribute interpreted_by* takes value from PROPERTY_OR_CLASS_INSTANCE resource defined in Section 2.2.
- *The attribute pref_attributes* associates an UNINTERPRETED_PREFERENCE. with list of PREFERENCE_URI identifier.

Example. In Table 5.7, an UNINTERPRETED_PREFERENCES (*#47*) correspond to set of property or class values of an ontology that are selected as being prefered is defined according to the addressed domain: *HotelIBIS* and *pizza margarita*

3.4 Context Based Preference Def nition

Context is a general term used to capture any information, that can be used to characterize the situations of an entity. The definition of preferences may depend on the context, where they are interpreted. For example, someone may interpret differently the cheap preference for the price of a hotel according to the location of the hotel (e.g. London, Paris).

Model. In our work, we don't address the study of context, but we provide with resource capable to refer to any explicit model of context. In our case, context is modeled as a set of multi dimensional attributes. To handle these preferences, the EXPRESS data model contains the resource presented on Table 5.8.

- *The attribute context_value* is a PROPERTY_OR_CLASS_INSTANCE Resource.

Table 5.8: Context_Based_Preference.

```
ENTITY CONTEXT_PREFERENCE_DEFINITION
   SUBTYPE OF (PREFERENCE);
   context_value:  PROPERTY_OR_CLASS_INSTANCE;
   preference:  PREFERENCE_ DEFINITION;
END_ENTITY;
```

```
#48= CONTEXT_PREFERENCE_DEFINITION (PROPERTY_VALUE('location',
'London'), (#110));
#49= CONTEXT_PREFERENCE_DEFINITION (PROPERTY_VALUE ('location',
'Paris'), (#120));
```

– *The attribute preference* associates with `Peference_Definition.`

Example. In this example, someone may interpret differently the same price range value for the price of a hotel if it is localized in *London* or *Paris*. Thus, `INTERVAL_PREFERENCE` definition is examined (e.g. #110 and #120).

```
#110= INTERVAL_PREFERENCE((#17), [(#3)]);
#120= INTERVAL_PREFERENCE((#17), [(#2)]);
```

In the first case #17 is used to define *expensive* preference.

```
#17= INTERVAL_VALUE(90,100);
#3= PREFERENCE_URI(101, 'expensive', 'cost');
```

But in the second case, same interval value (#17) is used to show *cheap* preference.

```
#17= INTERVAL_VALUE(90,100);
#2= PREFERENCE_URI(101, 'cheap', 'cost');
```

Finally, in Table 5.8 according to context value (*Paris* or *London*) the *cheap or expensive* preferences are interpreted differently.

4 Resource Preference Relationship

In the previous section we have described our proposed Preference Model. In this section we present the last part of our model definition consists of linking the Preference Model to the Ontology Model. The representation of an ontology requires a model of ontologies that provides the primitives needed to express the entities and relationships. It contains operators to deal with them. Many ontology models have been proposed in the literature [Kalinichenko et al., 2003], [Fankam et al., 2008]. They come from different fields and disciplines, the main ones being the description logics, the logic of frames and databases. For example,the ontology model PLIB is issued the database area, OWL and RDF-Schema

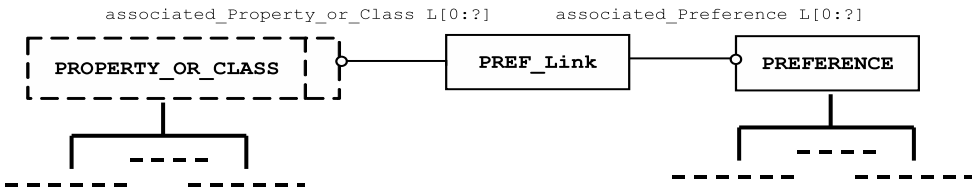


Figure 5.5: EXPRESS-G Representation of Preference Link Approach

models are respectively used from the description logics and semantic networks and F-Logic from the logic of frames.

Our approach associates any preference model to any ontology resource model that allows to manipulate the model of the ontology through its meta-model. Indeed, according to Figure 5.5, the `PREFERENCE` abstract entity of the preference model is associated to the `PROPERTY_OR_CLASS` resource entity. The `PROPERTY_OR_CLASS` resource represents data elements of the ontology model we refer to. It represents data and domain knowledge independently of any implementation model. Moreover, it is modeled independently from any ontology model (e.g. OWL, F-Logic, PLIB).

5 Conclusion

Users usually have varying preferences for the non-functional criteria depending on the situation they find themselves in, and of course different requestors will have different preferences. A good mechanism should not only allow to express values for each property, but preferably also represent the relations among the preferences. For example, a user may consider the price property as more important than location when requesting a reservation service (I prefer cheap hotel than high quality). Hence, the selection approach needs to provide mechanisms for users to specify their preferences, that is which of the non-functional properties they feel more strongly about and also relations between these properties.

As we discussed in this chapter, it is difficult to predict how many non-functional properties will be available, and additionally the type of these properties. For example, the evaluation function to compute the price criteria will be very different from the function to calculate the location criteria. It is very difficult to define a universal evaluation function for all kinds of non-functional properties. Non-functional properties exhibit constraint over the functionality. These properties involve qualitative or quantitative features. A model for non-functional properties, that can be used in preference descriptions (cheap, expensive, near...) is required. Due to the versatility of non-functional properties (and the fact that new ones might be required at any time) it is unlikely that a complete standard set can be identified. Furthermore, criteria should differ depending on the domain.

In this chapter designed the formal, shared and generic preference model is designed. Set up for handling preference for ontological data is realized. In this model firstly, the ontology's instances are taken into account by referring to their corresponding ontology's entities. Then the preference model has been inspired by the ones defined. This model is composed of several kinds of preferences identified in the literature. These preferences are independent of any logical model of data. They are defined a posteriori by providing the interpretation of each preference, if required. The generic characteristic of this preference model is provided by the capability to define a relationship with any ontology of a given

domain. And finally, the Preference Link is introduced. This link is used to attach preference model to the ontology model.

This model has been formally formalized in the EXPRESS data modeling technique. A set of preferences validating this proposal has been defined. The Eco Toolkit has been used to operationally validate this model. The whole EXPRESS model together with a validation file of EXPRESS instances are given in Appendix A.

In the next chapter, we focus on the integration of our preference model in ontology based databases. The OntoDB OBDB is used for this propose and the OntoQL language is used for manipulating and querying the resulting database model and data.

Chapter 6

Extending Ontology Based Databases with Preferences

Contents

Abstract. The purpose of this chapter is to propose an extension to an existing OBDB, called OntoDB in order to support semantic description of preferences. In this chapter, firstly we describe how our proposed Preference Model can be attached to an ontology model through the manipulation of the Meta-model level. Then, how a specific Preference Model is linked to the concept of class or property of the ontology model is shown. Finally, how we can use these information to query OBDB with preferences is described.

1 Introduction

In the previous chapter, we have proposed a model to represent user preferences and show how this model can be related to ontologies and ontological data is shown. In this chapter, an extension of an existing OBDB, called OntoDB, through the extension of its ontology model in order to support semantic description of preferences is proposed.

In Chapter 4 Section 2, the main elements of the OntoDB architecture has been presented. In the same chapter the OntoQL language with an Ontology and Data definition language (`CREATE, ALTER, DROP clauses`),a manipulation language (`INSERT INTO, DELETE, UPDATE clauses`) and query language (`SELECT`) is also described.

In this chapter, both OntoDB and OntoQL will be used to show how preferences can be handled in OBDB. To extend the OntoDB architecture with preferences, we focused on the metaschema (2) and ontology (4) parts. In the following sections, we present the proposed extension for both parties. In section 6.2 handling and querying preferences in OntoDB are described. Then we will present how the OntoQL Language is used to generate this extension. finally, in section 6.4 we define the OntoQL `PREFERRING` operator extending the query language parts.

2 Handling and Querying Preferences in OntoDB

OntoDB offers the necessary resources to store both the ontological model and the preference model in the same infrastructure. Thanks to the possibility to instantiate the meta-schema part and to the use of the OntoQL language, the manipulation of these different components becomes possible. In this section, we first explain ontology representation in OntoDB then show how the OntoDB architecture is extended with preferences.

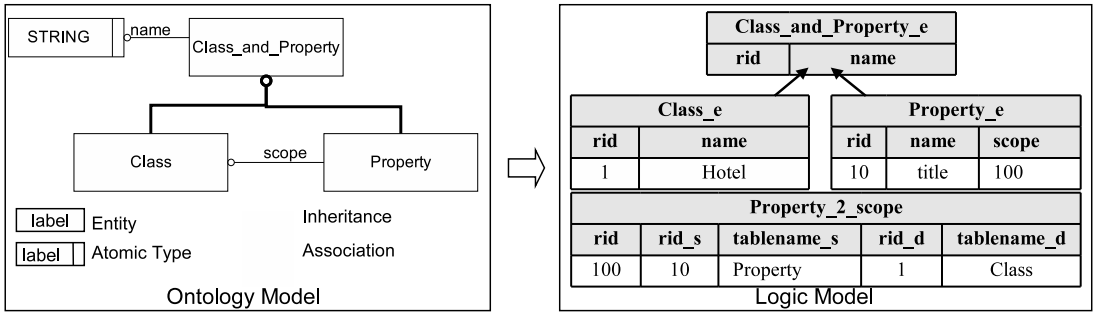


Figure 6.1: Ontology and Logic Model on OntoDB

2.1 Ontology Representation in OntoDB

In OntoDB, tables and columns are named by using the internal identifiers of classes and properties. On the left side of the Figure 6.1 a simple ontology model is represented with EXPRESS. This ontology model consists of three entities. The entity `Class_and_Property` has a single attribute *name* . This attribute indicates the name of a class or property. `Class_and_Property` is the super-entity Entity for

`Class` and `Property.`It is used to represent the classes and properties of ontologies. Both entities are linked by the association, to represent domain (a class) area of a property.

On the right hand side of Figure 6.1, the logic model used by OntoDB for storing ontologies is presented. A table is created for each entity of the ontology model. Each table possesses the attribute rid providing internal identification of database classes and properties. According to the hierarchy of entities in the model of ontologies, the `Class_e Property_e` table inherits Class_and_Property_e tables. Finally, to represent the association *scope*, the switch table `Property _2_scope` was created. It makes the link between a property and a class. The identifier of this property (by class) and the table where it is stored are shown in columns `rid_s` and `tablename_s` .

To give a sample of ontology representation, we use an *Hotel ontology*. This ontology is illustrated by using OntoQL syntax in Table 6.1.

Table 6.1: Hotel Ontology.

```
CREATE #Class Hotel(
DESCRIPTOR ( #name[fr] = 'hôtel'),
#property(
  name STRING,
  city STRING,
  starRate STRING));
```

```
CREATE EXTENT OF Hotel (
(name, city, starRate);
```

```
INSERT INTO  Hotel (name, city, starRate)
VALUES ('HotelFormule1', 'Paris', 2);
INSERT INTO Hotel (name, city, starRate)
VALUES ('HotelIBIS', 'Paris', 3);
INSERT INTO Hotel (name, city, starRate)
VALUES ('HotelMercure', 'Paris', 5);
```

Table 6.1 shows the statement to create the Hotel Class. The `DESCRIPTOR` clause is used to describe this class. It is assumed that French (FR) and English (EN) are the only natural language used to describe the concepts and institutions of the Hotel Ontology. Then, `#Property` clause is used to create properties together with the class. The `CREATE EXTENT OF` statement is used to define the extension of class (i.e., the set of properties used to describe instances of Hotel).

Table 6.2: Hotel Ontology Instances.

oid	name	city	starRate	price
23	HotelFormule1	Paris	2	30
28	HotelIBIS	Paris	3	85
29	HotelIBIS	Poitiers	3	65

Example. In the example *name, city starRate and price* attributes are instantiated according to their primitive types (e.g. `STRING` and `integer (INT)).` In the Table 6.2 the representation of *Hotel Ontology* instances are shown with their internal identifier `(oid)` .

2.2 OntoDB Extension with Preferences

The extension of OntoDB to handle the preferences consists in describing a set of CREATE OntoQL clauses that extend the ontology model with preferences. The symbol (#) is used to precise that the creation is at the entity Meta-model level. For example, creating a preference concept at the ontology model level is performed by adding another new concept in the ENTITY Meta-model resource using the following OntoQL clause.

```
CREATE ENTITY #Preference (
oid int,
...
);
```

In this section the creation process of all the entities of the Preference Model, which is presented in Chapter 5, is shown.

2.2.1 Preference_URI

In our model each preference is associated with PREFERENCE_URI to give a characterization to a preference with a set of attributes. These attributes are described in Section 5.1.

Table 6.3: Preference_URI.

```
CREATE ENTITY #Preference_URI(
  #code INT,
  #name STRING,
  #classification STRING);
```

```
INSERT INTO #Preference_URI (
#code, #name, #classification)
VALUES (100,'cheap','cost');
INSERT INTO #Preference_URI(
(#code, #name, #classification)
VALUES (50,'low level','quality');
```

Model. In Table 6.3 the creation of the Preference_URI entity is described with OntoQL specifications.

Table 6.4: Preference_URI Instances.

oid	code	name	classification
1316	100	cheap	cost
1315	101	expensive	cost
1319	40	lux	StarRate

Example. This entity has *code, name and classification* attributes assigned to internal *oid* in the Data Part of the OntoDB architecture (Table 6.4).

2.2.2 Numeric_Preference

Numeric preferences are described in Chapter 5 Section 3.2.2. This type of preferences are interpreted by numeric values.

Table 6.5: Numeric_Preference.

```
CREATE ENTITY #Numeric_Preference(
UNDER #Interpreted_Preference(
  #number_value INT,
  #REF(#Preference_URI)));
```

```
INSERT INTO #Numeric_Preference(
#number_value, #REF(#Preference_URI)
VALUES (5, 'http://....');
```

Model. In the Table 6.5 the syntax of the `Numeric_Preference` creation is given. This preference is created under `Interpreted_Preference.`

Example. Example values are inserted into *number_value* and *URI* attributes. URI attribute takes its value from the related `PREF_URI` adress. So, *lux hotel* takes the value of star rating which is *'5'*.

2.2.3 Interval_Preference

Interval preferences are interpreted by interval values which is described in Chapter 5 section 3.2.3.

Table 6.6: Interval_Preference.

```
CREATE ENTITY #Interval_Preference(
UNDER #Interpreted_Preference(
  #min_value INT,
  #max_value INT,
  #REF(#Preference_URI)));
```

```
INSERT INTO #Interval_Preference(
#min_value, #max_value, #REF(#Preference_URI))
VALUES (90, 100, 'http://....');
```

Model. In Table 6.6 OntoQL statement is used to create `Interval_ Preference` with *min_value*, *max_value* and *URI* attributes.

Example. In the same table, the interval [90..100] is attached to the *expensive* cost preference. It is shown in Table 6.6.

2.2.4 Boolean_Preference

Boolean preferences that are explained in Chapter 5 Section 3.2.5 are interpreted as the presence or the absence of a given feature.

Table 6.7: Boolean_Preference.

```
CREATE ENTITY #Boolean_Preference(
UNDER #Interpreted_Preference(
  #properties (STRING)ARRAY,
  #REF(#Preference_URI));
```

```
INSERT INTO #Boolean_Preference(
#properties, #REF(#Preference_URI))
VALUES (ARRAY['casino','golf'], 'http://....');
```

Model. In the Table 6.7 the creation of the entity Boolean Preference is presented.

Example. In the example a number of characteristics of a hotel (e.g. facilities) are taken into consideration like *Internet access, room service, room with a view or facilities* for the physically disabled. This example shows only *casino* and *golf* facilities are attached to `Preference_URI` *lux*.

2.2.5 Enumerated_Preference

Enumerated Preferences show the population preferences in a set. This type of preferences are explained in Chapter 5 Section 3.2.1

Table 6.8: Enumereted_Preference.

```
CREATE ENTITY #Enumerated_Preference(
 UNDER #Interpreted_Preference(
 #pref_values REF(STRING) ARRAY)
  #REF(#Preference_URI);
```

```
INSERT INTO #Enumerated_Preference(
#properties, #REF(#Preference_URI))
VALUES (ARRAY['HotelFormule1, HotelPrimier'], 'http://....');
```

Model. The OntoQL syntax of the creation `Enumerated_Preference` is given in Table 6.8. This entity takes set of values from `Property_or_Class_Instance` resource. In the OntoDB architecture this resource is shown as a `String` definition.

Example. In the example `Preference_URI` *cheap* is attached to a set of hotel instances with *oid* shown in Table 6.4.

2.2.6 Fuzzy_Preference

Fuzzy Preferences associate probability values to a given preference. It is described in Chapter 5 Section 3.2.4.

Table 6.9: Fuzzy_Preference.

```
CREATE ENTITY #Fuzzy_Preference(
UNDER #Interpreted_Preference(
 #prob_value FLOAT,
 #properties REF(STRING)ARRAY)
  #REF(#Preference_URI));
```

```
INSERT INTO #Fuzzy_Preference (
#prob_value, #properties, #REF(#Preference_URI))
VALUES (0.95, ARRAY['casino','jakuzi'], 'http://....');
```

Model. In Table 6.9 OntoQL statement is used to create Fuzzy Preference with *prob_value* and *URI* attributes.

Example. In the example, probability value 0.95 is attached to *casino and jakuzi* property values to describe *lux* hotel with 'http://....'.

2.3 Linking Ontologies and Preferences at the Ontology Model Level

Once the preferences and the ontology have been defined, the next step consists in linking the ontological entities to the preferences that are expressed on these entities. In OntoDB the preferences containers are defined at the meta schema level. They need to be linked to their corresponding ontological entity at the meta schema as well.

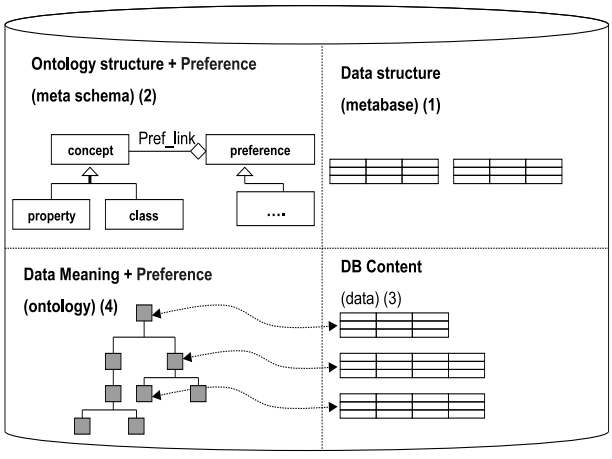


Figure 6.2: Extended OntoDB Architecture.

In Figure 6.2 extended OntoDB architecture is presented. After Preference Model is created with OntoQL statements, used link it to the ontology model using `Pref_Link` association. The following

OntoQL statement, that is **ALTER** clause is used to create such a link in OntoDB. An aggregate of preferences encoded in the attribute, `#pref_link` is added to the `#Property_or_Class` entity.

```
ALTER ENTITY #concept ADD ATTRIBUTE #PREF_Link
REF (#Preference) ARRAY
```

We have shown how to create the *Hotel Ontology* at the preferences. Now we show how to link preferences to class of the ontology. We used an **UPDATE** clause is used to attach `Preference_URI` to `#Property_Or_Class_Instance` (any instance value of the domain ontology).

Table 6.10: UPDATE Clause.

```
UPDATE #class
set #PREF_LINK = ARRAY['cheap','standard', 'full board']
WHERE  #name = 'Hotel'
```

In the Table 6.10 the preference *cheap* is attached to the **class Hotel**. Here, the names are used for readability but identifiers are in fact used.

3 Preference-driven Query Processing in OntoDB

The ability to model preferences and exploit preferential information to assist users in searching for items has become an important issue in knowledge representation. The user of an information system rarely knows exactly what he is looking for, but once shown a piece of information he can quickly tell whether it is what he needs.

Preferences can be regarded as special kind of soft filter expressions [Kissling 2002]. They take a set of values and refine this set. In a similar fashion to the way that Kießling extended SQL to enable database querying with preferences, [Siberski, Pan, and Thaden 2006] presents an extension to SPARQL to query ontological information with preferences.

In this section our objective is to describe semantic queries that handle preferences expressed at the semantic level in OntoDB architecture. It is presented as a comprehensive extension of OntoQL query language which directly supports the expression of preferences. This includes formal syntax and semantics of preference expressions for OntoQL. An implementation of the querying proposed here have been completed based on the OntoQL query engine [Jean, 2007] and building on the implementation of [Siberski et al., 2006] where the iterative processing of preference querying is performed as a solution modifier (similarly to the way the classical sorting functionality `FILTER` is done).

Our fundamental idea here is similar; a new query clause is introduced to allow the construction of preferences as soft constraints. This new query clause is called as `PREFERRING`.

3.1 Syntax of Preferring Operator

OntoQL is based on SQL. The syntax of OntoQL query language is shown below:

```
<query specification> ::= <select clause>
<from clause> [ <where clause> ]
[ <group by clause>] [ <having clause> ] [ <order by clause> ]
[ <namespace clause> ] [ <language clause> ]
```

The extended syntax of OntoQL with PREFERRING operator is given below:

```
<query specification> ::= <select clause>
<from clause> [ <where clause> ]
[ <group by clause> ] [ <having clause> ]
[ <order by clause> ] [preferring clause]
[ <namespace clause> ] [ <language clause> ]
<preferring clause> ::= <boolean preferenceIdentifier expression>
<boolean preferenceIdentifier expression> ::= <boolean term>
| <boolean value expression> OR <boolean term>
<boolean term> ::= <boolean factor>
| <boolean term> AND <boolean factor>
<boolean factor> ::= [ NOT ] preferenceIdentifier
<preferenceIdentifier> ::= <identifier>
```

In the Preference Model each Preference_URI must be thought as a <preferenceIdentifier>. Because of this when using more than one <preferenceIdentifier>, boolean operators like AND, OR, NOT should be used.

3.2 Query Interpretation

In order to handle the preferences in the OntoQL queries, a preference interpreter has been developed on top of the OntoQL engine. This is materialized by adding a PREFERRING clause in the OntoQL SELECT clause. First, we show same samples of the proposed syntax.

```
SELECT 'selection'
FROM 'tableReference'
PREFERRING 'preferenceIdentifer'
```

```
SELECT 'selection'
FROM 'tableReference'
PREFERRING 'preferenceIdentifer' AND 'preferenceIdentifer'
```

```
SELECT 'selection'
FROM 'tableReference'
PREFERRING NOT 'preferenceIdentifer'
```

```
SELECT 'selection'
FROM 'tableReference'
PREFERRING 'preferenceIdentifer' OR 'preferenceIdentifer'


SELECT 'selection'
FROM 'tableReference'
PREFERRING NOT 'preferenceIdentifer' AND 'preferenceIdentifer'
```

Two example queries, based on the situations presented above, are presented below.

Example Query-1 User's preference that is related with the cost of the hotel, is specified indirectly by the term *cheap*. Running this query through OntoQL will have exactly the behavior where any type of hotel will be considered equal without examining its relationship with the target concept.

List me cheap hotels.

```
SELECT name, price
FROM Hotel
PREFERRING 'cheap';
```

Example Query-2 User is looking for a hotel which has *cheap price and lux level equipment* in the room.

```
SELECT 'selection'
FROM 'tableReference'
PREFERRING 'cheap' AND 'lux';
```

3.3 SPARQL Interpretation

According to the SPARQL language interpretation the preference is directly specified in FILTER clause.

In this example for the *cheap* property, which is on property "price" and whose interval is [45 .. 60], can be written using OntoQL as seen below.

```
SELECT name, price, description
FROM Hotel
WHERE price BETWEEN 45 and 60
```

The previous query can also be written in SPARQL language:

```
SELECT ?name ?price ?description
WHERE { ?h rdf:type Hotel . ?h name ?name .
?h price ?price . ?h description ?description }
```

This query will be rewritten using the FILTER clause of SPARQL:

```
SELECT ?name ?price ?description
WHERE { ?h rdf:type Hotel . ?h name ?name .
?h price ?price . ?h description ?description
FILTER(?price >= 45 && ?price <= 60) }
```

4 Conclusion

This chapter has presented an extension of a database architecture in order to handle preference modeling and querying with preferences not at the database logical model but at the semantic level offered by the ontology. This extension requires,

- the explicit representation of the ontology in the database. As a consequence, we have been able to attach the preferences to the classes and to the properties of the ontology and not to the columns of the logical model of the database, where instances or data are stored;

- the possibility to access and to manipulate the ontology model through the access and manipulation to the meta-model;

- the availability of an exploitation language allowing to manipulate the instances, their classes and the meta-model in the case of ontologies.

Our approach is implemented by extending the OntoDB system. OntoDB offers the necessary facilities to store both ontologies and the preference model. Thanks to the use of the OntoQL language, the manipulation of these different components becomes easier.

As a consequence, semantic queries that handle preferences, expressed at the semantic level have been described. And thus they are abstracted from the logical model in the same infrastructure. The usage of `PREFERRING` operator is to use user preferences as `PREFERENCE_URI` 's in the query sentences. For OntoQL query language this operator is firstly used in OBDBs research area.

In the next chapter we will describe an case studies of the proposed extensions.

Chapter 7

Case Studies

Contents

Abstract. The whole model and operational resources have been defined in Chapter 5 and Chapter 6. Now it is possible to show, how our approach on a case study works. The following section illustrates it with an example taken from the tourism domain and specifically an online holiday booking system.

1 Introduction

In order to illustrate our approach, let us consider a research scientist, Marc, who wants to book a hotel room for his summer holiday. To do this, he uses his favorite online holiday booking system. This booking system manages a set of hotels disseminated over the world. These hotels offer various leisure facilities and are rated according to the international hotels rating standard ranging from *1 star hotels* to *5 star hotels*. The number of stars depends on various characteristics including the level of comfort and the leisure facilities offered by the hotel. The price of a room depends on the category of the hotel and of the period. The booking system uses an ontology about the tourism domain. This ontology formally describes the knowledge about the domain.

Marc would like to minimise the budget, he will spend, for his holiday. He prefers living in cheap hotels and has a preference for standard room facilities. Regarding this situation, the objective is to satisfy Marc making sure that his preferences are considered. Two preferences are identified here: the room facilities ***'standard'*** *and the cost description* ***'cheap'****.*

2 Case Study 1 - Handling Preferences in Ontology

The instantiation of our model to this case study is described below. First a fragment of the tourism ontology is given, the expressed preferences as instances of our preference model on chapter 5 are described. Notice that, the instances are shown in two ways: one as `EXPRESS` instances and the other as triples, to show their possible multiple representations.

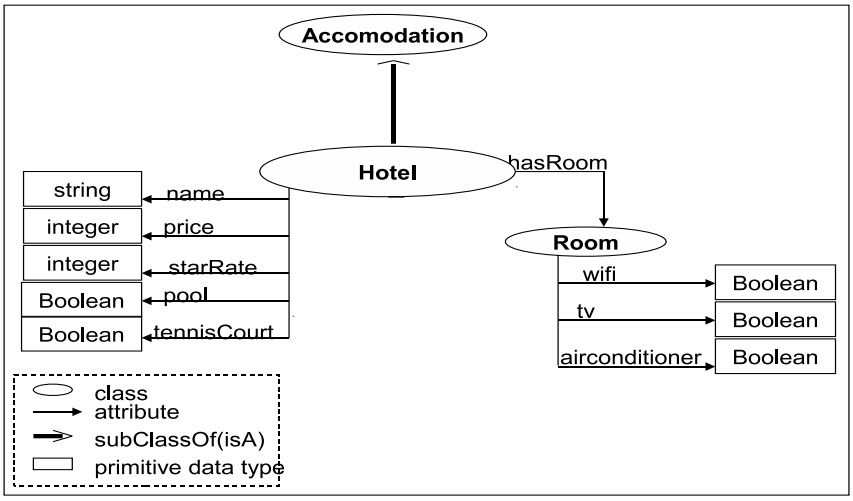


Figure 7.1: Tourism Ontology Concepts.

2.1 The Domain Ontology: A Vacation Ontology Instantiation

The Tourism Ontology defines the concepts, describing the accommodation and complementary services usually offered by hotels. It defines rooms, characterized as single, double or suite. It also defines the

Table 7.1: Ontology Instantiation with EXPRESS.

#id= **name**(tv, wifi, tenniscourt, pool, airconditioner, starRate, price);
#50= **HotelFormule1**(tv, airconditioner, 2, 30);
#51= **HotelKyriad**(tv, wifi, airconditioner, 3, 55);
#52= **HotelIBIS**(tv, wifi, airconditioner, 4, 75);
#53= **HotelMercure**(tv, wifi, airconditioner, tennisCourt, pool, 5, 95);

concept reservation and additional services such as room facilities (e.g., TV, wifi), provision of meals (e.g., half board), etc. The main concepts of this ontology are given in Figure 7.1 and Table 7.1 also Figure 7.2 shows ontology instantiation part.

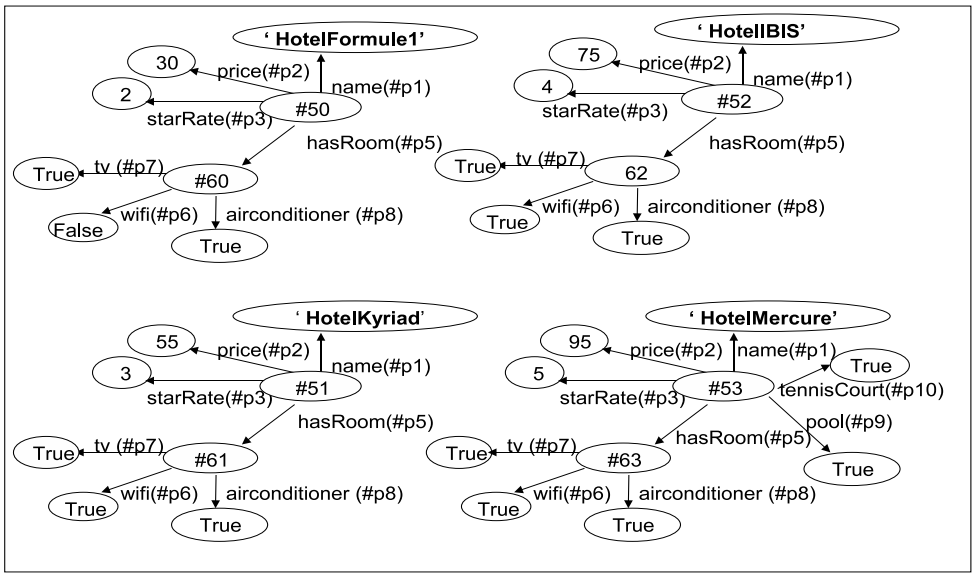


Figure 7.2: Tourism Ontology Instantiation.

2.2 Preference Model Instantiation

The preference model defines different type of preferences. In this case study, it is instantiated to interpret *standard* and *cheap* preferences respectively as Numeric and Interval Preference types. Their corresponding `PREFERENCE_URI` definitions are defined in Table 7.2 and Table 7.4. These two types of preferences are examined with different values.

Firstly for *cheap* preference in the scenario preference identifiers, dependent to *cost* category, are formed.

Table 7.2: Preference_URI Examples.

#1=**Preference_URI**(99,’very_cheap’,’cost’);
#2=**Preference_URI**(100,’cheap’,’cost’);
#3=**Preference_URI**(101,’expensive’,’cost’);
#4=**Preference_URI**(102,’very_expensive’,’cost’);

Then **INTERVAL_PREFERENCE** is defined. This preference type helds interval values (with min and max values). They are formed for PREFERENCE_URI 's, covering intervals like *cheap, expensive and very expensive*, that will be used within a pricing example. (Tablo 7.3).

Table 7.3: Interval Preference Example.

–Interval Values.
#11= **Interval_Value**(20, 45);
#12= **Interval_Value**(45, 60);
#13= **Interval_Value**(60, 90);
#17= **Interval_Value**(90, 100);
–Interval Preference examples [20..45],...,[90..100].
95= **Interval_Preference** (#11, [#1]);
#100= **Interval_Preference** (#12, [#2]);
#105= **Interval_Preference** (#13, [#3]);
#110= **Interval_Preference** (#17, [#4]);

In Table 7.3's first part, instances examples for different interval values are presented. In the second, related interval values and PREFERENCE_URI values presented in Table 7.2 place. Instances, that are numbered as (#100, #105 and #110) are examples for INTERVAL_PREFERENCE types.

For *standard* preference usage example, that takes place on second case in scenario, star ratings of hotels, which is used for quality standards, PREFERENCE_URI connected to (*StarRate*) category are formed (Tablo 7.4).

Table 7.4: Preference_URI Examples.

#7=**Preference_URI**(50,'low','starRating');
#8=**Preference_URI**(51,'standard','starRating');
#9=**Preference_URI**(52,'middle','starRating');
#10=**Preference_URI**(53,'lux','starRating');

NUMERIC_PREFERENCE is defined. This preference type is used for description of numerically expressable preferences. Numeric preference is matched with PREFERENCE_URI for different star rating values which are illustrated in Table 7.5.

Table 7.5: Numeric Preference Examples.

–Numeric Preferences
#26=**Numeric_Preference**(2, [#7]);
#27=**Numeric_Preference**(3, [#8]);
#28=**Numeric_Preference**(4, [#9]);
#29=**Numeric_Preference**(5, [#10]);

Instances of PREFERENCE_URI' s with numbers #26, #27, #28 ve #29, which are used widely for different star ratings, are illustrated as examples of NUMERIC_PREFERENCE .

In the next part two preference types, presented above, will be matched with ontology model in the OntoDB architecture .

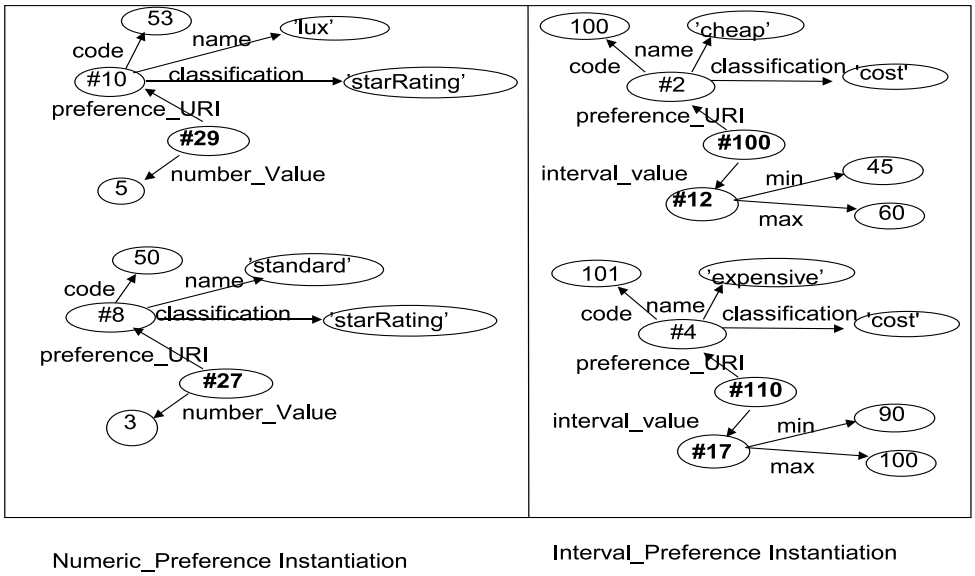


Figure 7.3: Tourism Ontology Instantiation.

2.3 Ontology Preference Link

The set of links between the ontology instances (#50, #51, #52, #53) and the preferences (#26, #27, #28, #29, #95, #100, #105, #110) are defined by providing the set of instances of the PREFERENCE_LINK express entity presented in Chapter 5 Section 4. The set of preferences are defined in Table 7.6.

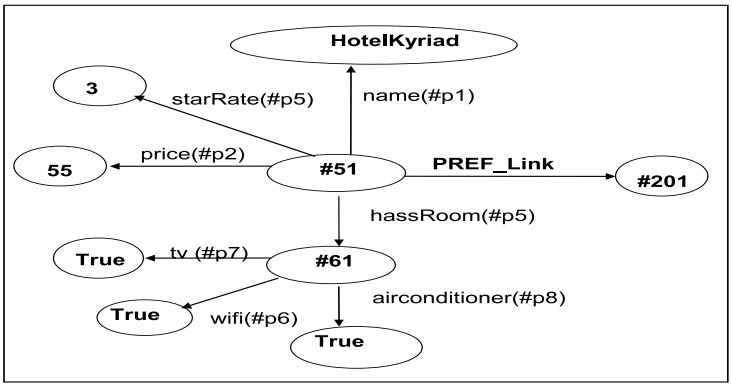


Figure 7.4: Preference Link Instantiation.

Table 7.6: Preference Link Instantiation.

Examples:
#201=**PREF_Link**(#51,[#27,#100]);

Explication. For example, #201=**PREF_Link**(#51, (#27, #100)) is expressed that the NUMERIC_PREFERENCE and INTERVAL_PREFERENCE are attached to the *HotelKyriad* in Figure 7.4 .

3 Case Study-2: Preference Based Querying in OntoDB

The ontology, described previously, is defined in the OntoDB system by populating its different entities. Table 7.7 gives the OntoQL fragment that creates `Hotel` class.

Table 7.7: Create Ontology.

Domain Ontology:

```
CREATE Class Hotel(
  #id int,
  #name String,
  #starRate int,
  #price int,
  #airCond boolean,
  #tv boolean,
  #wifi boolean,
  #pool boolean,
  #jakuzi boolean);
```

```
CREATE EXTENT OF Hotel (
  id, name, starRate, price, airCond, tv,
  wifi, pool, jakuzi, tennisCourt ,casino);
```

In order to define ontology instances, the `INSERT INTO` OntoQL clause is used. Table 7.8 gives the ONTOQL fragment that instantiates into OntoDB the previously defined Hotel class with the **Hotel Kyriad** and **Hotel Formule1**.

Table 7.8: Ontology Instantiation.

Preference Instantiation Example:

```
INSERT INTO Hotel (id, name, starRate, price,
airCond, tv, wifi, pool, jakuzi, tennisCourt, casino)
VALUES (51, 'HotelFormule1', 2, 30, 'true', 'true','false', 'false', 'false', 'false', 'false');
INSERT INTO Hotel (id, name, starRate, price,
airCond, tv, wifi, pool, jakuzi, tennisCourt, casino)
VALUES (51, 'HotelKyriad ', 3, 55, 'true', 'true', 'true', 'false', 'false', 'false', 'false');
```

3.1 Extension of the OntoDB with Preferences

The preference model defines different types of preferences. In our case study, it is instantiated for *numeric* and *interval* preference types to encode respectively the *standard* and *cheap* preferences like graphically represented on Figure 7.6. For our illustration, on defining the preferences attached to the *star rating* and to the *price* attributes are focused.

In Table 7.9 following OntoQL statement describes the creation of `PREFERENCE` , `PREFERENCE_URI` , `Numeric_Preference` and `Interval_Preference` entities. And every created entity has its own

Table 7.9: Preference Model Instantiation.

```
CREATE ENTITY (#Preference_URI(
  #code INT,
  #name STRING
  #classification STRING));
```

```
  INSERT INTO #Preference_URI)(
 #code, #name, #classification STRING));
 VALUES(100, 'cheap','cost');
```

```
CREATE ENTITY #Preference(
  #oid INT,
  #URI REF(#Preference_URI));
```

```
  INSERT INTO #Preference)(
 #oid, #URI REF(#Preference_URI)));
 VALUES(1001, 1319);
```

```
CREATE ENTITY #Numeric_Preference(
UNDER #Interpreted_Preference(
  #number_value INT,
  #URI REF(#Preference_URI)));
```

```
INSERT INTO #Numeric_Preference(
#number_value, #URI REF(#Preference_URI))
VALUES(2, 1319);
```

```
CREATE ENTITY #Interval_Preference(
UNDER #Interpreted_Preference(
  #min_value INT,
  #max_value INT,
  #URI REF(#Preference_URI)));
```

```
INSERT INTO #Interval_Preference(
#min_value, #max_value, #URI REF(#Preference_URI)
VALUES(45, 60, 1319);
```

`oid` . For `PREFERENCE_URI` entity this `oid` is called as `#URI.` This attribute is used in `Numeric_Preference` and `Interval_Preference` .

When the preferences and the ontologies are defined, it is possible to link ontology classes to the preferences that are expressed on these classes. For this purpose, the manipulated ontology class is augmented by preferences with the use of the `ALTER` clause (Chapter 6 Section 2.3).

Table 7.10: Preference Link.

```
UPDATE #class set #Pref_link=ARRAY ['cheap','standard','full board']
WHERE where name='Hotel';
```

A preference is attached to an instance of a hotel. For example, the preference cheap is attached to the Kyriad hotel using the following `UPDATE` OntoQL clause. Here names are put for readability purposes, but in practice identifiers are used.

Table 7.11: Update Class Hotel ADD Preference.

name	price	star Rating	PREFERENCE (cost,quality,promotion)
Kyriad	55	3	[cheap,standard,full board]

The database can be queried, when the preferences to the ontology concepts are linked.

3.2 Querying OntoDB with Preferences

Once the three previous steps are realised, it becomes possible to query the implemented model. Two examples of queries, with an asserted quality on the data with the use of PREFERRING clause, are given below.

```
SELECT name, starRating, preference
FROM Hotel
PREFERRING 'standard';
```

```
SELECT p.#name, pref.#number_value
FROM #property p,
       unnest(p.#preference) as pref
 where pref.#label='standard';
```

```
SELECT name, starRating, preference
FROM Hotel
WHERE starRating < 3;
```

name	starRating	preference
Hotel Kyriad	3	standard

Figure 7.5: Query for Numeric Preference.

```
SELECT name, price, preference
FROM Hotel
PREFERRING 'cheap';
```

```
SELECT p.#name,
pref.#min_value,pref.#max_value
FROM #property p,
       unnest(p.#preference) as pref
 where pref.#label='cheap';
```

```
SELECT name, price, preference
FROM Hotel
WHERE price BETWEEN 45 and 60;
```

name	price	preference
Hotel Kyriad	55	cheap

Figure 7.6: Query for Interval Preference.

Notice that all the authorised subclauses of the SELECT OntoQL clause can be used in building the query. The last clause is the PREFERRING clause used for rewriting the queries into standard OntoQL queries. This extension preserves upward compatibility with the classical SELECT clause. As in the previous example, the query has to be rewritten according to type of preference in the PREFERRING

clause. In this case, the preference cheap has been defined as an interval preference attached to the property price. It takes its values in the interval of [45, 60] which can be interpreted by the clause `BETWEEN`.Thus, the query is automatically rewritten as follows. As standard is defined as a numeric preference attached to the property `starRate` having the value 3, when the `PREFERRING` clause is interpreted, the query is automatically rewritten.

4 Conclusion

Scenerio established in this study presents examples derived from Tourism Domain. As described in the Case Study-1, meta-models themselves do not directly contribute to the development of information systems, but are the starting point for developing methodically well-founded approaches for information systems. In this case Preference Meta-model design principles are presented.

The implications of an attempt to use this meta-model on OntoDB, which are the development of a specification language (symbols for the model constructs of the meta model) and guidelines on their use, are demonstrated in Case Study-2.

In most existing systems, constraints and preferences are implicit in selections made by the customer. Most of the search engines support only the specification of hard constraints, i.e. solutions are only shown if all constraints are matched. If a customer specifies a lot of search preferences, existing online booking engines will often return no solution. This is called the empty-result-effect. Often the entire search process has to be repeated over and over again in order to manually find a compromise, causing a very tedious and frustrating search.

In our solution for Case Study-2 preferences are defined by the Preference Meta-model, which takes place as a generic side model in the OntoDB architecture, explicitly. For querying the domain knowledge a `PREFERRING` operator is implemented. Using this operator a novel preference query is executed (as in Figure 6.4) and then, with rewriting this query in SQL and SPARQL, the interpretation procedures of these languages are examined.

Finally, to increase users' precision in queries, `PREFERRING` operator is used with preference identifiers together.

Conclusion and Future Works

In this thesis in order to represent user's preferences, we proposed a modular, sharable and generic model of preferences. We have described how the OntoDB ontology based database framework has been extended in order to handle preferences at the semantic level instead of handling them on the logical level of the data. Our model is composed of several types of preferences usually addressed in the literature in a separate way. These preferences are independent of any logical model of data.

Our contributions are twofold. Firstly, we showed, how preferences model can be attached to an ontology and be manipulated on the meta-model level. Then, we generated semantic queries that handle preferences expressed at the semantic level. We believe that the possibility to access the meta-model level well adapted to define model extensions that preserve upward compatibility with the extended model.

Preference Meta Model

Handling preferences has been addressed in various information systems research areas. By analysing the literature, we note that the notion of preferences as considered in the databases domain is introduced mainly at the logical level. Preferences are defined on top of the manipulated models themselves and consequently, they depend on the model they extend. In the Semantic Web domain, even if an ontology is used to define preferences, the approaches are static and not enough flexible. It is generic to handle different preference models and are hardly adaptable to other contexts. This is because these approaches handle mostly preferences at ontology's instance level. For the storage issue, any of the approaches provide a real storage possibility of preferences model. Some of them hard code the models in the application (e.g. (Siberski et al, 2006)). None of the previously mentioned approaches offer simultaneously the possibility to define preferences at the semantic level to allow their persistence and to provide a dedicated language for querying with preferences facilities.

To overcome these drawbacks, we propose an approach of preferences management by introducing a data model, which is abstracted from the concerned database logical model. Our approach consists in associating any preference model to any ontology resource model that allows to manipulate the model of the ontology through its meta-model. The ontology's instances are taken into account in the preference model by referring to their corresponding ontology's entities. Preferences can be expressed on property or class instances.

In this work three distinct elements, detailed below, compose the model: i) the ontology model resource; ii) the preference model; iii) and finally the link between these 2 resources.

Firstly, to summarize our ontology-based data representation, ontology model resource is created. The main goal of ontologies is the representation of the semantic of objects of a given domain. This goal is reached by assigning objects to ontological classes and by describing them using ontological properties.

Secondly, we compiled different definition of the preferences found in the literature. Each preference is associated with a set of attributes that give a characterisation for this preference. Notice that the `PPREFERENCE_URI` entity is separately defined and could be interpreted from a more general knowledge model. Our model introduced specific resources allowing to define preferences. It was created in a hierarchical manner that represents different preference types together and it is possible to extend this model. Therefore, new preference types can be added to the model and integration with other present models is possible, like fuzzy approaches or context based modeling approaches.

Finally, Preference Link is established between Domain Ontologies and Preference Model. When all types of preferences in our preferences model have been defined, we need to link them to the ontological model. To do this, we use the ontology preference relationship. This relationship allows to attache a particular preference to a given ontological entity, being either a class or a property. It represents both the data and the domain knowledge (described in an ontology) independently of any implementation model. Moreover, it is modelled independently from any specific model of ontology (e.g. OWL, F-Logic).

The most important contribution of this research thesis is to establish a Meta-Model, that is independent from the Domain Ontologies. Also attachment of Domain Ontologies' values, such as Instance and Data Values, to preference types was accomplished.

OntoDB and Preference Meta-Model

Our approach is implemented by extending the OntoDB (OBDB) system. OntoDB's architecture has strong OBDB similarities with the architecture of MOF (Meta Object Facility) metadata [Kobryn, 1999]. This architecture consists of four layers. Layer model M1 MOF architecture is OntoDB's conceptual model, subset of the ontology. The meta-model layer corresponds to the model M2 ontology, the layer meta-meta-model M3 (MOF model) is the meta-model of language definition of the ontology. It is reflexive. This architecture allows us to integrate automatically [L.Bellatreche and Dehainsala, 2004], to migrate and exchange bodies not necessarily defined in the ontology model. It allows us to use the results of work performed under the MOF.

OntoDB uses its own data model for its ontologies. Such ontologies are represented as sets of instances of an object schema (often called meta-model) expressed in a particular modelling formalism (XML-schema for OIL et OWL, EXPRESS for PLIB). As a result, this representation provides an exchange format for these ontologies (an XML document for OWL, a physical file of EXPRESS instances for PLIB). In OntoDB, the ontology schema is generated automatically from EXPRESS model (for OWL for instance, the OWL model only needs to be expressed in EXPRESS). The multi-instantiation and property subsumption (subproperty) specific features of OWL are included. It is quite unique and able to keep track of almost all the sophisticated features of OWL, giving strong inference abilities. In such context,

The OntoDB model,

- uses a relational database on which the ontologies and their instances are stored.
- supports automatic integration and management of heterogeneous populations whose data, schemas and ontologies are loaded dynamically;
- offers data access, at the ontology level, whatever the type of the used DBMS (relational, object-relational or object oriented) is and
- supports evolutions of the used ontologies (adding new classes, new properties, etc.) and their population schemas.

Other contribution of this thesis is the integration of the Preference Model, that was established as an explicit side model to OntoDB. The meta-schema part of the OntoDB Meta-model was extended by adding Pref_Link attribute to the Property Table. Thus, layer M2 on the OntoDB architecture is extended with preferences by using EXPRESS language skills. Technically, this extension is possible only if the meta-model allows to describe the ontology model that can be manipulated.

Preference Based Querying

All the described OBDB approaches lack to offer primitives that are able to represent non-functional aspects related to the ontology models such as quality of service, preferences or security. Indeed, most of the well known ontology models including OWL and PLIB do not provide built-in constructors to represent these notions. To overcome the problem of handling non-functional characteristics, specific attributes (e.g., note, remark) are introduced or particular properties are defined. The advantage of this approach is the possibility to adapt non-functional descriptions to any ontology model keeping its definition unchanged and preserving upward compatibility. Technically, this extension is possible only if the meta-model, that describe the model of the ontology can be manipulated. Indeed, such an extension requires being able to attach any element of the model of the ontology to the model of the non-functional elements. However, expressing the personal queries by the non-functional properties (preference qualifiers), which are daily used, is an issue, that has not yet been addressed in research area of OBDB.

In this thesis the extension of the ontology model with the Preference Model permits to attach various types of preferences to the entities of the ontology. We have been able to describe semantic queries that handle preferences expressed in the semantic level, and thus abstracting from the logical model. The possibility to access the meta-model level is well adapted for defining some model extensions that preserve upward compatibility with the extended model.

Last contribution is the addition of the ability of the preference-based query to OntoQL language. Personal queries on OntoDB architecture were evaluated at semantic level with use of established Preference Model.

PREFERRING OPERATOR with OntoQL

In order to handle the preferences in the OntoQL queries, a preference interpreter has been developed on top of the OntoQL engine. Similar to the SPARQL language, that is used in Semantic Web area and SQL is used in database applications, preferences were expressed as user's limitation by qualifiers based on Preference Model in OntoQL query language. Thus, to query preferences for OntoDB architecture, `PREFERRING` operator was defined as a new operator to OntoQL query language. This is materialized by adding a `PREFERRING` subclause to OntoQL SELECT clause.

Query examples were implemented on two different chosen preference type. However, other examples directed to Context and Fuzzy preference types were not demonstrated in the context of this research thesis.

As demonstrated in the database research area, established Preference Operator should be developed in a manner that comprises definitions/terms of priority and superiority. But, an application in this direction was not presented in the context of this research thesis.

Established sample applications disconsider certain overlaps such as querying of distinct preference types together. Improvement of Preference Model is required to resolve overlaps, which were disconsidered in this research thesis.

Future Work

This work has opened several new directions and perspectives. We propose a model to exploit user preferences on domain knowledge. We plan to study how our preference model may express preference dependencies. Another challenging research direction is towards user models that combine multiple user aspects, e.g. preferences, abilities, demographic data etc. Models of increased expressive power such as those outlined above require advanced query personalization mechanisms and logic. For example, combining and reconciling information stored in diverse profiles, and profile hierarchies are challenging topics that need to be addressed. Finally, we are very interested in methods for the automatic construction of user profiles based on the preference model described in this work. Existing methods have mainly focused on construction of simple keyword profiles.

From the Data Warehousing perspective a possible improvement is to extend the Preference Model in order to take aggregation into account. This will allow users expressing their preferences directly at the query aggregation level too (Rizzi, 2007). From a performance point of view, the evaluation and optimisation of preference queries (e.g. cost based optimisation) and the complexity implications of introducing preferences into queries would be beneficial.

Also it is wanted to develop Pareto preference to show relative importance of personal preference criteria and prioritization queries to arrange preferences according to their priorities. Finally, top-k queries over uncertain data in the quantitative framework can also be a study.

Résumé

Résumé. La prise en compte des préférences utilisateurs a été proposée comme une solution au problème de l'accès efficace à la grande quantité de données manipulées par les applications. Cependant, les approches existantes définissent habituellement les préférences pour un domaine particulier. Ainsi, il est difficile de les partager et de les réutiliser dans d'autres contextes. Dans cet thèse, nous proposons un modèle de préférences générique et partageable. Le modèle intègre plusieurs types de préférences proposés dans la littérature mais traités de manière séparée. Notre approche, qui définit les préférences au niveau des ontologies qui décrivent la sémantique des données manipulées, offre un mécanisme de stockage du modèle de préférences et un langage pour l'interrogation avec les préférences. Elle est implémentée en utilisant une Base de Données à Base Ontologique (BDBO), étendue pour prendre en compte les préférences.

INTRODUCTION

Le développement et l'adoption rapide d'Internet et des nouvelles technologies de l'information rendent disponible une grande quantité de données gérées à travers différents systèmes d'information. En conséquence, lors d'une activité de recherche, les utilisateurs sont souvent submergés par les résultats fournis en réponse à leurs requêtes. De plus, parce que la plupart des systèmes d'information classiques ne prennent pas en compte les caractéristiques spécifiques de chaque utilisateur, les résultats d'une requête sont les mêmes quel que soit l'utilisateur qui la soumet. Bien que cette approche ait montré ses limites dans de nombreuses applications, la plupart des systèmes d'information ne prennent pas en compte dans leur conception la diversité des besoins des utilisateurs et la diversité de leurs préférences.

Les techniques de personnalisation sont une approche alternative pour résoudre ce problème et rendre aisé l'accès à la grande quantité de données traitées par les applications. En particulier, la prise en compte des préférences utilisateurs constitue un des fondements de cette approche. Les préférences expriment les souhaits des utilisateurs qui veulent trouver dans la réalité ce qui correspond le mieux à leur souhait.

Toutefois, modéliser des préférences est un problème difficile car elles sont par nature complexes, multiples, hétérogènes, changeantes voire contradictoires. De plus, elles sont difficiles à exploiter et doivent l'être selon le contexte dans lequel elles ont été définies. La modélisation des préférences et leur prise en compte pour la personnalisation de l'information ont été considérées dans plusieurs travaux de recherche dans les domaines des Bases de Données (BD), Entrepôts de Données (ED), du Web Sémantique (WS), de la Recherche d'Information (RI) et de l'Interface Homme Machine (IHM). Cependant, jusqu'à présent, le traitement des préférences est fortement couplé aux applications soit dans le code de l'application selon un modèle ad-hoc, soit au niveau de la base de données en fonction de son modèle logique, soit au niveau de l'IHM de l'application.

Ceci rend difficile le partage et la réutilisation de préférences entre applications. Pour permettre le partage et la réutilisation des préférences entre applications, notre approche consiste à les traiter au niveau des ontologies de domaine qui décrivent le ou les domaines abordés par ces applications. L'utilisation d'ontologies pour le développement d'applications a pour but d'expliciter la sémantique des données traitées par les applications afin notamment de faciliter l'interopérabilité. L'approche que nous proposons repose sur l'utilisation de bases de données à base ontologique (BDBO) qui permettent de gérer dans la même infrastructure aussi bien les données que les ontologies qui en explicitent la sémantique. Nous proposons d'étendre les BDBO pour pouvoir prendre en compte des préférences utilisateurs.

Même si l'idée du traitement des préférences au niveau sémantique n'est pas nouvelle, en particulier dans le contexte du WS, à notre connaissance aucune approche ne répond simultanément aux trois besoins suivants qui permettent une prise en compte complète des préférences : - définir un modèle précis pour représenter les préférences utilisateurs. Ce modèle doit permettre de représenter différents types de préférences et de prendre en compte le contexte dans lequel elles peuvent être définies; - proposer une solution de persistance pour stocker les préférences définies selon le modèle précédent. Cette solution de persistance doit intégrer le fait que le traitement des préférences est intéressant quand on gère un gros volume de données; - exploiter les préférences pour personnaliser l'accès aux données de l'application en utilisant un langage d'interrogation dédié.

L'approche présentée dans cet thèse répond à ces trois problèmes de la manière suivante : 1) propo-

sition d'un modèle de préférences basé principalement sur les types de préférences proposés dans les approches issues des communautés BD et WS. Ce modèle est formellement spécifié avec le langage EXPRESS [IS010303.02, 1994] de manière à ce que sa définition soit la plus précise possible; 2) extension de la BDBO OntoDB [Dehainsala et al., 2007b], [Dehainsala et al., 2007a] pour permettre de représenter les préférences définies. Nous avons choisi cette BDBO car elle offre des facilités pour étendre le modèle d'ontologies utilisé (e.g. OWL [Dean and Schreiber, 2004], PLIB [Pierra, 2003a]). Ceci nous a permis de lier le modèle de préférences au modèle d'ontologies utilisé; 3) extension du langage d'exploitation associé à OntoDB, c'est à dire OntoQL et SPARQL pour prendre en compte les préférences utilisateur. Ces préférences sont simplement spécifiées par un nom. Elles sont ensuite interprétées selon le type de préférence, ce qui permet de réécrire la requête de manière à ne retourner que les résultats pertinents pour l'utilisateur.

ÉTAT DE L'ART

Dans cette section, nous faisons une revue des approches de gestion des préférences utilisateurs telles qu'abordées dans les domaines des BD, du WS et des ED qui sont les plus liés à nos domaines d'intérêts. Notre but est d'étudier ces approches afin de proposer un modèle de préférences le plus générique possible.

Préférences et Bases de Données

La prise en compte des préférences a été abordée dans plusieurs travaux de recherche dans le domaine des Bases de Données ([Kiesling and Kostler, 2000], [Kiesling, 2002],[Chomicki, 2003], [Agrawal and Wimmers, 2000], [Koutrika and Ioannidis, 2004], [Viappiani et al., 2006]). Les préférences dans ce contexte sont définies au niveau du modèle logique, en particulier sur les valeurs des colonnes des tables. Selon le type de métrique, deux manières d'exprimer les préférences ont été proposées : l'approche qualitative et l'approche quantitative. Les approches qualitatives permettent à l'utilisateur de définir des préférences (relatives) entre les tuples de la base [Kiesling, 2002],[Chomicki, 2003]. Les préférences sont définies sur le contenu et à l'aide d'une relation binaire entre les tuples [Chomicki, 2003]. Si nous considérons deux tuples t1 et t2 par exemple, t1 > t2 signifie que l'utilisateur préfère le tuple t1 au tuple t2. Dans ce contexte, Kießling et Kostler proposent une approche qualitative (constructor approach) dans laquelle les préférences sont exprimées par un ordre partiel strict et sont définies de manière formelle à l'aide des formules logiques du premier ordre [Kiesling, 2002]. Les constructeurs définis sont intégrés au langage relationnel Preference SQL [Kiesling, 2002]. Par exemple, le constructeur Highest(c) exprime le fait que pour 2 tuples t1 et t2, on préfère le tuple ayant la valeur la plus élevée pour la colonne c. Cette approche est connue sous le nom de modèle de requête BMO [Kiesling, 2002]. Elle est similaire à celle mise en uvre par Chomicki à l'aide de l'opérateur Winnow [Chomicki, 2003]. Notons également que Skyline, introduit par Börzsönyi et al. [Borzsonyi et al., 2001] est une variante qui étend l'opérateur Winnow.

Les approches quantitatives permettent de définir des fonctions de score afin de calculer un score numérique, encore appelé préférence absolue, pour chaque tuple. Les résultats sont ordonnés selon ce score calculé. Dans ce contexte, Agrawal et Wimmers définissent les préférences en introduisant une

valeur préférée pour chaque colonne des tables de la base de données [Agrawal and Wimmers, 2000]. Par exemple, si nous considérons la table Film définie comme Filml(titre, prixMin, prixMax), la préférence <*, 20, 40 > indique que les films préférés sont ceux qui ont un prix de vente compris entre 20 et 40. La préférence ainsi définie est par la suite utilisée pour calculer un score entre 0 et 1 pour chaque film. Koutrika et Ioannidis introduisent quant à eux la notion de préférence atomique en spécifiant un ensemble de couples <condition, score> où condition est une condition sur les valeurs de colonnes et score est le degré d'intérêt entre 0 et 1 de cette condition [Koutrika and Ioannidis, 2004]. Les préférences atomiques peuvent être combinées et utilisées pour produire des préférences implicites. Par exemple, en considérant la même table Film, l'expression <Film.prixMax =40, 0.8> indique que le degré d'intérêt pour les films ayant un prix maximal de 40 est de 0.8. L'opérateur Top(k) a été introduit dans ce contexte [Koutrika and Ioannidis, 2004].

Préférences et Web Sémantique

La notion de préférence est également cruciale dans le domaine du Web Sémantique. Différents modèles ont été proposés pour représenter différents types de préférences [Siberski et al., 2006], [P. Gurský and Vaneková, 2008], [Toninelli et al., 2008].

Les travaux de Siberski et al. [Siberski et al., 2006] définissent une extension au langage SPARQL [Toninelli et al., 2008] permettant d'exprimer des préférences. Cette extension consiste en l'ajout au langage SPARQL de la clause PREFERRING. Deux types de préférences peuvent être définis. Les préférences booléennes sont exprimées à l'aide d'une condition booléenne. Les résultats qui satisfont cette condition sont préférés à ceux ne la respectant pas. Par exemple, la condition rating=excellent indique que l'on préfère les films ayant une excellente évaluation. Les préférences de score sont définies par une expression. Les résultats pour lesquels l'évaluation de l'expression conduit à la plus forte valeur sont préférés. Par exemple, l'expression LOWEST price indique que l'on préfère les films les moins chers.

Gurský et al. [P. Gurský and Vaneková, 2008] proposent un modèle de préférences locales basé sur les logiques floues. Des préférences locales sont d'abord définies à partir de propriétés. Elles sont ensuite composées pour former des préférences globales. Par exemple, la préférence globale good Movie(x) peut être définie par la composition des deux préférences locales good Rating(x) et recentMovie(x). Toninelli et al. [Toninelli et al., 2008] introduisent un modèle de préférences basé sur des ontologies mais qui cette fois-ci s'appuie sur les priorités. Par exemple, pour trouver les hôtels de grand standing, la qualité du service doit être une priorité.

Afin de personnaliser la recherche sur internet, Sieg et al. [Sieg et al., 2004] basent leur approche également sur une ontologie. Ils utilisent le contexte de l'utilisateur et re-ordonnent les résultats retournés pour chaque requête donnée. Ce contexte est représenté comme une instance d'une ontologie de domaine appelée ontologie de référence. Les concepts de cette ontologie sont annotés par des scores d'intérêts dérivés et mis-à-jour implicitement à partir du suivi du comportement de l'utilisateur. Cette représentation est appelée profil utilisateur ontologique.

Préférences et Entrepôts de Données

Il y a peu de travaux effectués dans le domaine des entrepôts de données comparés aux précédents. Bellatreche et al. [Bellatreche et al., 2005] et Mouloudi et al. [Mouloudi et al., 2006] ont traité le problème de la personnalisation pour les requêtes OLAP. Ils ont proposé pour cela une plateforme dans laquelle la possibilité de spécifier ses préférences et ses contraintes de visualisation pour l'affichage des résultats est donnée à l'utilisateur. Ses préférences peuvent être par exemple, la présence d'une dimension particulière des données d'un entrepôt. Les contraintes de visualisation représentent la taille du dispositif (PDA, téléphone portable, etc.) utilisé pour l'affichage du résultat d'une requête. Les auteurs présentent l'impact des préférences sur la conception physique d'un entrepôt (partitionnement des données, sélection des index et des vues matérialisées). Dans cette approche, aucun opérateur spécifique n'est proposé pour la prise en compte des préférences lors de l'interrogation. Notons également dans le cadre des Entrepôts de Données le travail de Ravat et al. [Ravat et al., 2007]. Ils proposent une approche de personnalisation d'un système décisionnel reposant sur une modélisation multidimensionnelle des données qui consiste à associer des poids aux différents composants d'un schéma multidimensionnel. Ils utilisent pour cela un langage de type ECA (Événement-Condition-Action).

Analyse de ces Approches

Nous avons résumé dans le tableau 1 les différentes approches présentées ci-dessus. Nous avons utilisé 4 critères pertinents pour évaluer ces approches. Le premier critère indique dans quel(s) domaine(s) (BD, WS ou ED) l'approche considérée est applicable. Le deuxième critère du tableau indique quel type d'approche est suivi (qualitative, quantitative, utilisation d'une ontologie, etc.). Le troisième critère caractérise les différents modèles proposés selon le niveau sur lequel le modèle de préférences est défini (physique, logique, sémantique). Le quatrième critère concerne l'existence d'un opérateur spécifique permettant la prise en compte des préférences utilisateurs lors de l'interrogation.

En analysant le tableau 1, nous notons que la notion de préférence, telle que prise en compte dans le domaine des BD, est introduite principalement au niveau logique des données. Par conséquent, les préférences ne peuvent être utilisées que dans le contexte particulier où elles ont été définies. Elles dépendent de la manière dont les données sont encodées ou implémentées dans le modèle logique. Dans le domaine du WS, même si une ontologie est utilisée pour définir les préférences, ces approches sont statiques et pas suffisamment flexibles et génériques pour prendre en compte des modèles de préférence différents (e.g. [P. Gurský and Vaneková, 2008]). Par ailleurs, elles sont difficilement adaptables dans d'autres contextes. Ces insuffisances sont souvent causées par le fait que ces approches définissent les préférences uniquement au niveau des concepts et instances de l'ontologie. Du point de vue du stockage, aucune des approches ne fournit un réel mécanisme de stockage du modèle de préférences, elles offrent plutôt la possibilité de stocker les préférences utilisateurs. Certaines codent le modèle directement dans l'application (e.g. [Siberski et al., 2006]).

De manière globale, aucune des approches proposées n'offre simultanément i) la possibilité de définir des préférences au niveau sémantique; ii) un mécanisme de persistance du modèle de préférences; iii) un langage dédié pour une interrogation avec la prise en compte des préférences.

Pour répondre à ces insuffisances, notre approche suggère d'abstraire la représentation logique des

Table 1: Récapitulatif des di fférentes approches de gestion des préférences

Auteur	Domaine	Approche (BD/WS/ED)	Niveau du Modèle	Opérateur de Requête de Préference
Kiebling (2002-2003)	BD	Qualitative	Logique	Preference SQL
Chomicki (2003)	BD	Qualitative	Logique	Winnow
Agrawal-Wimmers (2000)	BD	Quantitative	Logique	Non
Koutrica-Ionnidis (2006)	BD	Quantitative	Logique	Non
Siberski et al. (2006)	WS	Boolean-Scoring Preferences	Sémantique	SPARQL Clause Preferring
Sieg et al. (2007)	WS	Profil Utilisateur Ontologique	Sémantique	Non
Gurský et al. (2008)	WS	Probabiliste basé ontologies	Sémantique	Non
Tonielli et al. (2008)	WS	MiddleWare Meta Modèle	Semantic	Non
Bellatreche et al. (2005) Mouloudi et al. (2006)	ED	Qualitative	Logique	Non

données en liant le modèle de préférence que nous proposons à des ontologies. Elle repose sur l'utilisation d'une base de données à base ontologique (BDBO) étendue afin de prendre en compte les préférences.

Bases de Données à Base Ontologique : l'approche OntoDB

Les Bases de Données à Base Ontologique (BDBO) permettent de stocker conjointement les ontologies et les données qu'elles décrivent. Ces systèmes de persistance bénéficient des avantages des Bases de Données (par exemple, la scalabilité ou la gestion de la concurrence). OntoDB [Dehainsala et al., 2007b], [Dehainsala et al., 2007c] est une approche particulière qui propose d'introduire un méta-schéma dans l'architecture des BDBO afin de gérer à l'aide d'un méta-model réflexif le modèle de l'ontologie qui décrit la sémantique des données. Les 4 parties de l'architecture OntoDB sont décrites comme suit. La partie ontologie (4) permet de représenter complètement les ontologies de domaines. La partie méta-schéma (2) permet de représenter, au sein d'un modèle réflexif, à la fois le modèle d'ontologie utilisé et le méta-schéma lui-même. La partie méta-base (1) permet de représenter le schéma de représentation des objets du domaine couvert par les ontologies. Enfin la partie données (3) représente les objets du domaine; ceux ci sont décrits en termes d'une classe d'appartenance et d'un ensemble de valeurs de propriétés applicables à cette classe. Pour le schéma de l'ontologie, le méta-schéma joue le même rôle que celui du catalogue système dans les Bases de Données classiques. En effet, le méta-schéma supporte: i) un accès générique à l'ontologie; ii) l'évolution du modèle d'ontologie utilisée; le stockage de différents

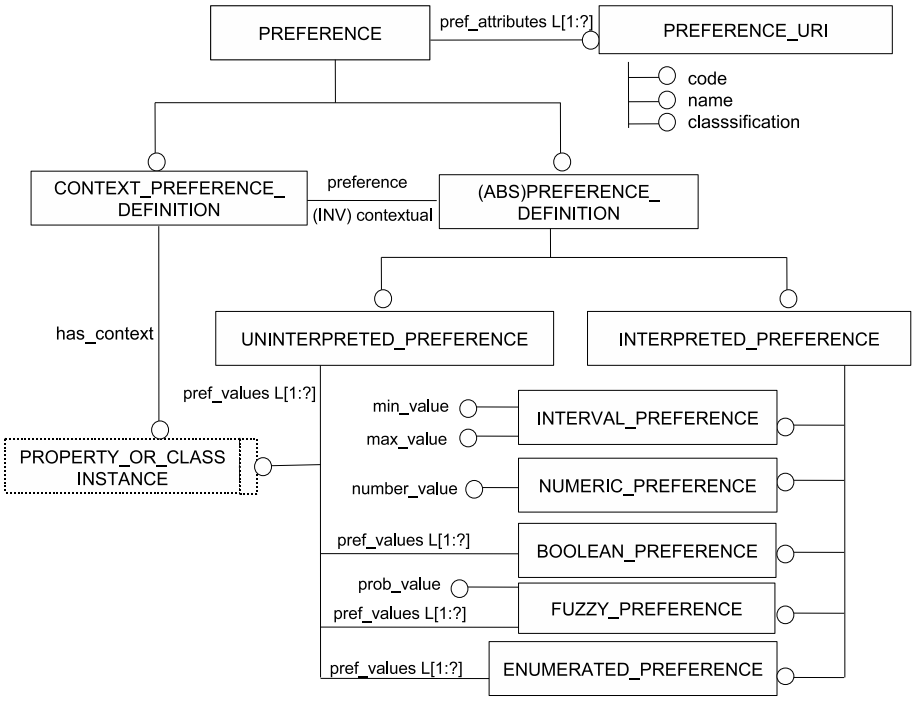


Figure 1: Vue d'ensemble du modèle de préférence

modèles d'ontologies.

Un langage d'interrogation appelé OntoQL [Jean, 2007], [Jean et al., 2005b] est associé à OntoDB. Il permet de manipuler les trois niveaux de l'architecture OntoDB: le méta-modèle (e.g., Class et Property), le modèle (e.g., Movie et Actor) et les instances (e.g., Scarface et Leonardo DiCaprio).

MODELE DE PREFERENCES BASE SUR UNE BDBO

Nous présentons ici les principaux composants du modèle et montrons comment il peut être intégré à OntoDB afin d'exprimer des préférences sur n'importe quel ensemble de données décrit sémantiquement par une ontologie.

Vue d'ensemble du modèle de préférences

Notre modèle de préférences est composé de trois éléments distincts, détaillés dans les sections suivantes: i) les ressources du modèle d'ontologies; ii) les différents types de préférence du modèle de préférences; iii) et enfin le lien entre le modèle de préférences et le modèle d'ontologies. La Figure 1 présente ces différents éléments. Dans la section suivante, nous détaillons les différents éléments de notre modèle de préférences.

Les ressources du modèle d'ontologies

Notre approche consiste à associer un modèle de préférences à n'importe quel modèle d'ontologies. Si nous considérons la figure 1 qui présente le modèle de préférences, l'entité Preference du modèle est associée à l'entité Property_Or_Class du modèle d'ontologies. Les instances de l'ontologie sont prises en compte dans le modèle de préférences en se référant à leurs entités correspondantes. Les préférences sont exprimées sur les instances de propriétés ou de classes. L'entité Property_Or_Class_Instance représente une instance de propriété ou une instance de classe de l'ontologie considérée.

Détail des éléments du modèle de préférences

La définition du modèle de préférences intègre différents modèles habituellement traités principalement dans les communautés BD et WS. Nous avons séparé les préférences qui sont génériques de celles qui dépendent du contexte où elles ont été définies. Les préférences peuvent être interprétables ou non. Nous désignons par préférences non interprétables, les préférences qui sont énumérées par un utilisateur ou un concepteur donné sans aucune fonction d'interprétation. Chaque préférence est associée à un ensemble d'attributs qui caractérisent cette préférence.

Préférences interprétées

Les préférences interprétées (Interpreted_Preference) sont des préférences associées à une procédure d'évaluation ou d'interprétation. Par exemple, la préférence recent(x) peut être interprétée comme étant releaseYear(x) > 2006. L'idée étant de définir des préférences associées aux types de données qui ont une relation d'ordre.

`Les préférences énumérées (Enumerated_Preference):` elles correspondent à l'énumération d'instances d'entités d'une ontologie qui sont préférées. Par exemple, une préférence pour les acteurs de films d'actions (actionActor) peut être définie comme étant (Actor(SylvesterStalone), Actor(WesleySnipes)). Cet ensemble exprime le fait que la préférence actionActor correspond à deux instances de film.

`Les préférences numériques (Numeric_Preference):` elles correspondent à des préférences qui sont interprétées par des valeurs numériques. Par exemple, la qualité d'un film peut être définie par la moyenne des évaluations qui lui sont associé (sur une échelle de 0 à 5). La relation d'ordre est celle définie sur les numériques.

`Les préférences booléennes (Boolean_Preference):` elles correspondent à des préférences associées à une liste de propriétés à valeurs booléennes dont on préfère que la valeur soit à vrai. Par exemple, on peut définir une préférence sur des films ayant obtenu un oscar.

`Les préférences de type intervalle (Interval_Preference):` elles correspondent à des préférences exprimées par une valeur minimale et maximale. Par exemple, les préférences OldMovie et RecentMovie peuvent être associées à la propriété releaseYear. Dans ce cas, la préférence OldMovie définit les dates de réalisation comprises dans l'intervalle [1970,2000] tandis que RecentMovie définit une date de réali-

sation comprise dans l'intervalle [2001,2009].

`Les préférences probabilistes (Fuzzy_Preference):` elles permettent d'exprimer des préférences à l'aide des valeurs de probabilité. Par exemple une préférence probabiliste peut être utilisée pour exprimer que la probabilité de préférer les films ayant une moyenne d'évaluation de 2, 3 ou 4 est respectivement 0.1, 0.2 et 0.7. Ceci permet de traiter des données ontologiques avec des approches issues des logiques floues.

Les préférences non interprétées

Les préférences non interprétées (Uninterpreted_Preference) correspondent à un ensemble d'instances de classes ou de propriétés d'une ontologie qui sont considérées comme préférées. Il n'y a pas de rationnel pour choisir ces instances. Par exemple si nous considérons le domaine du cinéma, Leonardo DiCaprio (instance d'acteur), Steven Spielberg (instance de réalisateur), romantique (instance de genre) représentent une préférence qu'un utilisateur peut exprimer dans le domaine du cinéma.

Les préférences dépendantes du contexte

Parfois la définition des préférences peut dépendre du contexte dans lequel elles sont interprétées. Par exemple, si nous considérons la réglementation liée à la classification des films, une préférence exprimée sur des films accessibles aux mineures peut varier d'un pays à un autre selon l'âge légal de la majorité. Dans ce cas, l'interprétation de la préférence dépend de la valeur d'une autre propriété (le pays de localisation dans notre cas).

Association du modèle d'ontologies aux préférences

Le dernier élément de notre modèle consiste en un lien entre le modèle de préférences et le modèle d'ontologies. Il s'agit alors d'établir le lien entre les classes et les propriétés du modèle d'ontologies et les préférences du modèle de préférences. Ces classes et propriétés sont modélisées à travers une entité nommée Property_Or_Class. Le lien est réalisé par une entité nommée PREF_Link (voir figure 1). Après la définition des différents éléments de notre modèle, la prochaine étape consiste à le stocker afin de faciliter son utilisation et son partage. Nous avons choisi comme modèle de stockage la BDBO OntoDB décrit dans la section 2.5. OntoDB a l'avantage de représenter le modèle d'ontologies utilisé. Cependant, comme les autres BDBO, OntoDB ne permet pas de représenter les préférences que notre modèle propose. Pour répondre à ce besoin, nous l'avons alors étendu. Notre extension de OntoDB a ainsi consisté à représenter notre modèle de préférences ainsi que le lien avec le modèle d'ontologies.

PRISE EN COMPTE DES PREFERENCES: EXTENSION DE LA BDBO OntoDB

Pour la prise en compte des préférences, nous devons créer dans OntoDB les entités nécessaires pour la gestion de notre modèle de préférences. Nous utilisons pour cela le langage OntoQL associé à OntoDB. Nous détaillons ci-dessous les différentes étapes.

Création des entités du modèle

Nous présentons tout d'abord l'exemple de création dans le système des deux entités PREFERENCE et Preference_URI à l'aide de la clause CREATE de OntoQL. Ensuite pour l'illustration de l'approche faite par la suite, nous prenons l'exemple des préférences Numeric_Preference et Interval_Preference qui sont créées directement sous l'entité PREFERENCE pour simplifier. Le symbole " # " signifie que ces entités sont créées au niveau méta-modèle de OntoDB.

```
CREATE ENTITY #Preference_URI(
              #code int, #name String, #classification String)

CREATE ENTITY #Preference(#id_pref int)

CREATE ENTITY #Numeric_Preference(
              UNDER #Interpreted_Preference(
              #number_value INT,
              #REF(#Preference_URI)));

CREATE ENTITY #Interval_Preference(
              UNDER #Interpreted_Preference(
              #min_value INT,
              #max_value INT,
              #REF(#Preference_URI)));
```

Matérialisation du lien entre modèle d'ontologies et modèle de préférences

Une fois que les entités principales pour les préférences ont été définies, nous devons les lier à leurs entités au niveau du modèle d'ontologies. Ce lien est représenté par l'entité PREF_Link dans la figure 1. Dans le système OntoDB, nous modifions l'entité Property_Or_Class du modèle d'ontologies. L'instruction OntoQL suivante est utilisée pour cette modification.

```
ALTER ENTITY #Property_Or_Class
ADD ATTRIBUTE #PREF_Link REF(#Preference) ARRAY
```

Cette instruction associe à chaque classe ou propriété de l'ontologie un ensemble (Array) de préférences.

Extension du langage OntoQL pour la prise en compte des préférences

La prise en compte des préférences dans les requêtes OntoQL est rendue possible grâce au développement d'un interpréteur pour les préférences. Ceci est matérialisé par l'introduction d'une sous clause PREFERING à la clause SELECT de OntoQL. Une fonction d'interprétation est associée à chaque type de préférence disponible dans notre modèle de préférences. La syntaxe de la clause SELECT ainsi qu'un exemple pour obtenir les films récents s'écrivent de la sorte :

```
SELECT 'selection'
FROM 'tableReference'
PREFERRING 'preferenceIdentifier'
```

CONCLUSION et PERSPECTIVES

Nous avons proposé dans cet thèse un modèle formel et générique pour la prise en compte des préférences utilisateurs. Nous avons décrit comment la base de données à base ontologique OntoDB et son langage associé ont été étendus pour gérer les préférences au niveau sémantique et non au niveau logique des données comme c'est le cas pour la plupart des approches, en particulier dans le domaine des bases de données. Notre modèle est indépendant de tout modèle logique de données. La possibilité de le lier à n'importe quel modèle d'ontologies le rend entièrement flexible. Ainsi, notre approche permet de définir des préférences au niveau sémantique, de stocker le modèle de préférences défini en offrant ainsi la possibilité de sa réutilisation. Enfin, il offre un langage d'interrogation prenant en compte ces préférences.

Ce travail ouvre plusieurs perspectives. Nous envisageons une évaluation conséquente de l'approche en considérant de manière complète les différents types de préférences proposées en utilisant des critères standards de Rappel/Précision. Nous envisageons également de coupler notre approche avec un modèle utilisateur dans le cadre des BDBO dont le travail est en cours. Par ailleurs, une amélioration possible de notre approche consiste à étendre le modèle de préférences afin de prendre en compte l'agrégation dans le cadre des entrepôts de données. Cette extension permettra aux utilisateurs de pouvoir exprimer leurs préférences également au niveau des requêtes d'agrégation. Une autre direction serait l'étude des procédés de fragmentation fondés sur les préférences.

Sur le plan de la performance, il serait intéressant de faire l'évaluation et l'optimisation des requêtes avec préférences (par exemple l'optimisation basée sur les coûts) et de mesurer les conséquences sur la complexité engendrée par l'introduction des préférences dans les requêtes.

Bibliography

[OMG, 2002] (2002). Metaobjectfacility(mof) specification v1.4. Technical report.

[Adomavicius and Tuzhilin,] Adomavicius, G. and Tuzhilin, A. *Knowledge and Data Engineering, IEEE Transactions on*, (6):734–749.

[Agrawal et al., 2001] Agrawal, R., Somani, A., and Xu, Y. (2001). Storage and Querying of E-Commerce Data. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 149–158.

[Agrawal and Wimmers, 2000] Agrawal, R. and Wimmers, E. L. (2000). A framework for expressing and combining preferences. In *SIGMOD Conference*, pages 297–306.

[Al and H., 2003] Al, S. and H., A. (2003). 'exploring the impact of customer empowerment on marketing strategy and information systems effectiveness. In *The 4th International Conference on Performance Measurement and Management*, pages 211–19.

[Alexaki et al., 2001] Alexaki, S., Christophides, V., Karvounarakis, G., and Tolle, K. (2001). Managing Voluminous RDF Description Bases. In *Proceedings of the 2nd International Workshop on the Semantic Web*, pages 1–13.

[Bellatreche et al., 2005] Bellatreche, L., Giacometti, A., Marcel, P., Mouloudi, H., and Laurent, D. (2005). A personalization framework for olap queries. In *DOLAP '05: Proceedings of the 8th ACM international workshop on Data warehousing and OLAP*, pages 9–18, New York, NY, USA. ACM.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):34–43.

[B.McBride, 2001] B.McBride (2001). Jena: Implementing the RDF Model and Syntax Specification. *Proceedings of the 2nd International Workshop on the Semantic Web*.

[Borzsonyi et al., 2001] Borzsonyi, S., Kossmann, D., and Stocker, K. (2001). The skyline operator. In *ICDE*, pages 421–430.

[Bozsak et al., 2002] Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., Stojanovic, N., Studer, R., Stumme, G., Sure, Y., Tane, J., Volz, R., and Zacharias, V. (2002). KAON - Towards a Large Scale Semantic Web. In *Proceedings of the 3rd International Conference on E-Commerce and Web Technologies (EC-WEB'02)*, pages 304–313, London, UK. Springer-Verlag.

[Brézillon, 2003] Brézillon, P. (2003). Focusing on context in human-centered computing. *IEEE Intelligent Systems*, 18(3):62–66.

[Brickley and Guha, 2004] Brickley, D. and Guha, R. V. (2004). Rdf vocabulary description language 1.0: Rdf schema. W3c recommenda-

tion, W3C. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[Broekstra et al., 2002] Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, pages 54–68.

[Brusilovsky, 2001] Brusilovsky, P. (2001). User modeling and user-adapted interaction. In *Adaptive Hypermedia*, pages 87–100.

[Buono et al., 2001] Buono, P., Costabile, M. F., Guida, S., Piccinno, A., and Tesoro, G. (2001). Integrating user data and collaborative filtering in a web recommendation system. In *In Proceedings of the Third Workshop on Adaptive Hypertext and Hypermedia*, pages 315–321. Springer Verlag.

[Carroll et al., 2004] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. (2004). Jena: Implementing the Semantic Web Recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters (WWW'04)*, pages 74–83, New York, NY, USA. ACM Press.

[Chaari et al., 2008] Chaari, S., Badr, Y., Biennier, F., BenAmar, C., and Favrel, J. (2008). Framework for web service selection based on non-functional properties. In *International Journal of Web Services Practices*, volume 3, pages 94–109.

[Cherniack et al., 2003] Cherniack, M., Galvez, E. F., Franklin, M. J., and Zdonik, S. B. (2003). Profile-driven cache management. In *ICDE*, pages 645–656. IEEE Computer Society.

[Chomicki, 2003] Chomicki, J. (2003). Preference formulas in relations queries. *ACM Transactions on Database Systems*, 28:1–39.

[Chong et al., 2005a] Chong, E. I., Das, S., Eadon, G., and Srinivasan, J. (2005a). An Efficient SQL-based RDF Querying Scheme. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, pages 1216–1227.

[Chong et al., 2005b] Chong, E. I., Das, S., Eadon, G., and Srinivasan, J. (2005b). An Efficient SQL-based RDF Querying Scheme. In *Proceedings of the 31st international conference on Very Large Data Bases (VLDB'05)*, pages 1216–1227.

[Damiani et al., 2001] Damiani, E., Oliboni, B., and Quintarelli, E. (2001). Modeling users' navigation history.

[Dean and Schreiber, 2004] Dean, M. and Schreiber, G. (2004). *OWL Web Ontology Language Reference*. World Wide Web Consortium. `http://www.w3.org/TR/owl-ref`.

[Dehainsala, 2007] Dehainsala, H. (2007). *Explicitation de la sémantique dans les bases de données : Le modèle OntoDB de bases de données à base ontologique*. PhD thesis, LISI/ENSMA et Université de Poitiers.

[Dehainsala et al., 2007a] Dehainsala, H., Pierra, G., and Bellatreche, L. (2007a). OntoDB: An Ontology-Based Database for Data Intensive Applications. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, pages 497–508.

[Dehainsala et al., 2007b] Dehainsala, H., Pierra, G., and Bellatreche, L. (2007b). OntoDB: An Ontology-Based Database for Data Intensive Applications. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, volume 4443 of *Lecture Notes in Computer Science*, pages 497–508. Springer.

[Dehainsala et al., 2007c] Dehainsala, H., Pierra, G., Bellatreche, L., and Aït-Ameur, Y. (2007c). Conception de bases de données à partir d'ontologies de domaine : Application aux bases de données du domaine technique. In *Actes des 1ère Journées Francophones sur les Ontologies (JFO'07)*, pages 215–230.

[Djuric et al., 2003] Djuric, D., Gasevic, D., and Devedzic, V. (2003). A mda-based approach to the ontology definition metamodel.

[Dragan Gasevic, 2006] Dragan Gasevic, Dragan Djruric, V. D. (2006). *The MDA-Based Ontology Infrastructure*.

[Dubois et al., 1998] Dubois, D., Prade, H., and Sabbadin, R. (1998). Qualitative decision theory with sugeno integrals. In *in: Proc. 14th Conf. on Uncertainty in Arti cial Intelligence*, pages 121–128. Physica Verlag.

[Fankam et al., 2008] Fankam, C., Jean, S., and Pierra, G. (2008). Numeric reasoning in the semantic web. In *SeMMA*, pages 84–103.

[Fischer, 2001] Fischer, G. (2001). User modeling in human-computer interaction. In *User Modeling and User-Adapted Interaction*, pages 65–68.

[Fishburn, 1988] Fishburn, P. (1988). *Nonlinear preference and utility theory*. Johns Hopkins University Press, Baltimore.

[Fuhr et al., 1999] Fuhr, N., Rittberger, M., and Christa, W.-H. (1999). *Information Retrieval [Elektronische Ressource] : Materialien zur Herbstschule*. Bonn.

[Gerhard et al., 1997] Gerhard, C. T., Thomas, C. G., and Fischer, G. (1997). Using agents to personalize the web. In *In Proc. ACM IUI'97*, pages 53–60. ACM Press.

[Giacomo and Lenzerini, 1995] Giacomo, G. D. and Lenzerini, M. (1995). What's in an aggregate: Foundations for description logics with tuples and sets. In *IJCAI (1)*, pages 801–807.

[Gruber and Olsen, 1994] Gruber, T. and Olsen, G. (1994). An ontology for engineering mathematics. In *Fourth Int'l Conf. Principles of Knowledge Representation and Reasoning*, pages 258–269.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220.

[H. K. Bhargava and Herrick, 1997] H. K. Bhargava, S. S. and Herrick, C. (1997). Beyond spreadsheets: Tools for building decision support systems. In *IEEE*, pages 31–39.

[H. Stefan and Kiesling, 2003] H. Stefan, E. M. and Kiesling, W. (2003). Preference mining: A novel approach on mining user preferences for personalized applications.

[Hansson, 2001] Hansson, S. O. (2001). Preference logic. *Handbook of Philosophical Logic (2 edition)*, 2:319–394.

[IS010303.02, 1994] IS010303.02 (1994). Product data representation and exchange - part 2: Express reference manual. *ISO-055*.

[ISO13584, 1998] ISO13584 (1998). Industrial automation systems and integration – Parts library – Part 42: Description methodology: Methodology for structuring parts families. Technical report, International Standards Organization, Genève.

[Jean, 2007] Jean, S. (2007). *OntoQL, un langage d'exploitation des bases de données à base ontologique*. PhD thesis, LISI/ENSMA et Université de Poitiers.

[Jean et al., 2006a] Jean, S., Aït-Ameur, Y., and Pierra, G. (2006a). Querying Ontology Based Database Using OntoQL. In *Proceedings of On the Move to Meaningful Internet Systems 2006:(ODBASE'06)*, volume 4275 of *Lecture Notes in Computer Science*. Springer.

[Jean et al., 2006b] Jean, S., Aït-Ameur, Y., and Pierra, G. (2006b). Querying Ontology Based Database Using OntoQL (an Ontology Query Language). In *Proceedings of On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences (ODBASE'06)*, volume 4275 of *Lecture Notes in Computer Science*, pages 704–721. Springer.

[Jean et al., 2005a] Jean, S., Pierra, G., and Ait-Ameur, Y. (2005a). OntoQL: an exploitation language for OBDBs. In *Proceedings of the VLDB 2005 PhD Workshop. Co-located with the 31th International Conference on Very Large Data Bases (VLDB'05)*, pages 41–45.

[Jean et al., 2005b] Jean, S., Pierra, G., and Ait-ameur, Y. (2005b). Ontoql: an exploitation language for obdbs. In *VLDB Ph.D. Workshop*.

[Jorstad et al., 2006] Jorstad, I., Thanh, D. V., and Dustdar, S. (2006). Personalisation of next generation mobile services. In *UMICS*.

[Kalinichenko et al., 2003] Kalinichenko, L., Missikoff, M., Schiappelli, F., and Skvortsov, N. (2003). Ontological modeling. In *Proceedings of the 5th Russian Conference on Digital Libraries RCDL2003*.

[Kiesling, 2002] Kiesling, W. (2002). Foundations of preferences in database systems. In Mars, N. J. I., editor, *Knowledge and Data Engineering*, pages 311–322. IOS Press, Amsterdam.

[Kiesling and Kostler, 2000] Kiesling, W. and Kostler, G. (2000). Preference sql -design, implementation, experience. In Mars, N. J. I., editor, *Knowledge and Data Engineering*, pages 778 – 789. IOS Press, Amsterdam.

[Klusch et al., 2003] Klusch, M., Ossowski, S., Omicini, A., and Laamanen, H., editors (2003). *Cooperative Information Agents VII, 7th International Workshop, CIA 2003, Helsinki, Finland, August 27-29, 2003, Proceedings*, volume 2782 of *Lecture Notes in Computer Science*. Springer.

[Kobryn, 1999] Kobryn, C. (1999). Uml 2001 : A standardization odyssey. In *Communications of the ACM, vol. 42, no. 10*.

[Kobsa, 2001] Kobsa, A. (2001). Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11(1-2):49–63.

[Koutrika and Ioannidis, 2004] Koutrika, G. and Ioannidis, Y. E. (2004). Personalization of queries in database systems. In *ICDE*, pages 597–608.

[Lacroix and Lavency, 1987] Lacroix, M. and Lavency, P. (1987). Preferences; putting more knowledge into queries. In Stocker, P. M., Kent, W., and Hammersley, P., editors, *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, pages 217–225. Morgan Kaufmann.

[L.Bellatreche and Dehainsala, 2004] L.Bellatreche, G.Pierra, D. X. and Dehainsala, H. (2004). Integration de sources de données autonomes par articulations à priori d'ontologies. In *INFORSID 2004*.

[Manber et al., 2000] Manber, U., Patel, A., and Robison, J. (2000). The business of personalization: Experience with personalization of yahoo! *Communications of the ACM*, 43(8):35–39.

[Mobasher et al., 2000] Mobasher, B., Dai, H., Luo, T., Sun, Y., and Zhu, J. (2000). Integrating web usage and content mining for more effective personalization. In *In Ecommerce and Web Technologies Lecture notes in computer Sciense (LNCS) 1875*. Springer-Verlag.

[Mostafa et al., 1996] Mostafa, M., Mukhopadhyay, S., Mostafa, J., Palakal, M., Lam, W., Xue,

L., and Hudli, A. (1996). An adaptive multi-level information filtering system. In *Proceedings of The Fifth International Conference on User Modeling*, pages 21–28.

[Mouloudi et al., 2006] Mouloudi, H., Bellatreche, L., Giacometti, A., and Marcel, P. (2006). Personalization of mdx queries. In Laurent, D., editor, *BDA*.

[P. Gurský and Vaneková, 2008] P. Gurský, T.Horváth, J. J. and Vaneková, V. (2008). User preference web search. experiments with a system connecting web and user. In *To appear in the Computing and Informatics Journal*, pages 25–32.

[Park et al., 2007] Park, M. J., Lee, J. H., Lee, C. H., Lin, J., Serres, O., and Chung, C. W. (2007). An efficient and scalable management of ontology. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, pages 975–980.

[Petrini and Risch, 2007] Petrini, J. and Risch, T. (2007). Semantic Web Abridged Relational Databases. In *Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA'07)*, pages 455–459.

[Pierra, 2003a] Pierra, G. (2003a). Context-explication in conceptual ontologies: the plib approach. In *ISPE CE*, pages 243–253.

[Pierra, 2003b] Pierra, G. (2003b). Context-Explication in Conceptual Ontologies: The PLIB Approach. In Jardim-Gonçalves, R., Cha, J., and Steiger-Garçao, A., editors, *Proceedings of the 10th ISPE International Conference on Concurrent Engineering (CE'03)*, pages 243–254.

[Pierra et al., 2005] Pierra, G., Dehainsala, H., Aït-Ameur, Y., and Bellatreche, L. (2005). Base de Données à Base Ontologique : principes et mise en œuvre. *Ingénierie des Systèmes d'Information*, 10(2):91–115.

[Prassas, 2001] Prassas, G.; Pramataris, K. P. O. D. G. (2001). Dynamic recommendations in internet retailing. In *ECIS 2001*, pages 27–28.

[Prud'hommeaux and Seaborne, 2006] Prud'hommeaux, E. and Seaborne, A. (2006). SPARQL Query Language for RDF. *W3C Candidate Recommendation 14 June 2007*. `http://www.w3.org/TR/rdf-sparql-query/`.

[Ravat et al., 2007] Ravat, F., Teste, O., and Zurfluh, G. (2007). Personnalisation de bases de données multidimensionnelles. In *Congrès Informatique des Organisations et Systèmes d'Information et de Décision (INFORSID), Perros-Guirec, 22/05/2007-25/05/2007*, pages 121–136, http://inforsid.irit.fr. INFORSID.

[Sabata et al., 1997] Sabata, B., Chatterjee, S., Davis, M., Sydir, J. J., and Lawrence, T. F. (1997). Taxonomy for qos specifications.

[Schenck and Wilson, 1994] Schenck, D. and Wilson, P. R. (1994). *Information Modeling the EXPRESS Way*. Oxford University Press.

[Schenk and Wilson, 1994] Schenk, D. and Wilson, P. (1994). *Information Modelling The EXPRESS Way*. Oxford University Press.

[Siberski et al., 2006] Siberski, W., Pan, J. Z., and Thaden, U. (2006). Querying the semantic web with preferences. In *In Proceedings of the 5th International Semantic Web Conference (ISWC*, pages 612–624.

[Sieg et al., 2004] Sieg, A., Mobasher, B., and Burke, R. (2004). Inferring user's information context: Integrating user profiles and concept hierarchies. presented at 2004 Meeting of the International Federation of Classification Societies.

[Staehli et al., 1995] Staehli, R., Walpole, J., and Maier, D. (1995). Quality of service specification for multimedia presentations. *Multimedia Systems*, 3:251–263.

[Theobald et al., 2004] Theobald, M., Weikum, G., and Schenkel, R. (2004). Top-k query evaluation with probabilistic guarantees. In *In VLDB*, pages 648–659.

[Toninelli et al., 2008] Toninelli, A., Corradi, A., and Montanari, R. (2008). Semantic-based discovery to support mobile context-aware service access. *Computer Communications*, 31(5):935–949.

[U. Shardanand, 1995] U. Shardanand, P. M. (1995). Social information filtering: Algorithms for automating 'word of mouth'. In *In Proceedings of the Conference on Human Factors in Computing Systems (CHI95)*, pages 210–217.

[Viappiani et al., 2006] Viappiani, P., Faltings, B., and Pu, P. (2006). Preference-based search using example-critiquing with suggestions. *Journal of Artificial Intelligence Research*, 27.

[Weld et al., 2003] Weld, D. S., Anderson, C., Domingos, P., Etzioni, O., Gajos, K., Lau, T., and Wolfman, S. (2003). Automatically personalizing user interfaces. In *In IJCAI03*, pages 1613–1619.

Appendix A

Preference Model with Express Language

Express Code

```
SCHEMA PREFERENCE_MODEL;

ENTITY PREFERENCE
ABSTRACT SUPERTYPE OF (ONEOF (PREFERENCE_DEFINITION, CONTEXT_PREFERENCE_DEFINITION));
END_ENTITY;

ENTITY PREFERENCE_DEFINITION
ABSTRACT SUPERTYPE OF (ONEOF (UNINTERPRETED_PREFERENCE, INTERPRETED_PREFERENCE));
END_ENTITY;

ENTITY PREFERENCE_URI;
CODE: INTEGER;
NAME:STRING;
CLASSIFICATION:STRING;
END_ENTITY;

ENTITY INTERPRETED_PREFERENCE
ABSTRACT SUPERTYPE OF (ONEOF (BOOLEAN_PREFERENCE,ENUMERATED_PREFERENCE,NUMERIC_PREFERENCE,INT
END_ENTITY;

ENTITY ENUMERATED_PREFERENCE
SUBTYPE OF (INTERPRETED_PREFERENCE);
        PREF_VALUES: LIST[1:?] OF PROPERTY_OR_CLASS_INSTANCE;
        PREF_ATTRIBUTES: LIST[1:?] OF PREFERENCE_URI;
END_ENTITY;

ENTITY NUMERIC_PREFERENCE
```

```
SUBTYPE OF (INTERPRETED_PREFERENCE);
        INTERPRETED_BY: NUMBER_VALUE;
        PREF_ATTRIBUTES: LIST[1:?] OF PREFERENCE_URI;
END_ENTITY;

TYPE NUMBER_VALUE= NUMBER;
END_TYPE;

ENTITY BOOLEAN_PREFERENCE
SUBTYPE OF (INTERPRETED_PREFERENCE);
        INTERPRETED_BY: LIST[1:?] OF PROPERTY_VALUE;
        PREF_ATTRIBUTES: LIST[1:?] OF PREFERENCE_URI;
END_ENTITY;


ENTITY INTERVAL_PREFERENCE
SUBTYPE OF (INTERPRETED_PREFERENCE);
INTERPRETED_BY: INTERVAL_VALUE;
PREF_ATTRIBUTES: LIST [1:?] OF PREFERENCE_URI;
END_ENTITY;

ENTITY INTERVAL_VALUE;
min_value: REAL;
max_value: REAL;
WHERE min_value < max_value;
END_ENTITY;

ENTITY  FUZZY_PREFERENCE
SUBTYPE OF (INTERPRETED_PREFERENCE);
  INTERPRETED_BY: PROB_VALUE;
  PREF_VALUES: LIST[1:?] OF PROPERTY_VALUE;
  PREF_ATTRIBUTES: LIST[1:?] OF PREFERENCE_URI;
END_ENTITY;

TYPE PROB_VALUE= REAL;
WHERE ((SELF>0) AND (SELF<1));
END_TYPE;

ENTITY UNINTERPRETED_PREFERENCE
SUBTYPE OF (PREFERENCE_DEFINITION);
INTERPRETED_BY: LIST[1:?] OF PROPERTY_OR_CLASS_INSTANCE;
    PREF_ATTRIBUTES: LIST[1:?] OF PREFERENCE_URI;
END_ENTITY;
```

```
ENTITY CONTEXT_PREFERENCE_DEFINITION
SUBTYPE OF (PREFERENCE);
CONTEXT_VALUE: PROPERTY_OR_CLASS_INSTANCE;
PREFERENCE: PREFERENCE_DEFINITION;
END_ENTITY;

TYPE PROPERTY_OR_CLASS_INSTANCE=
SELECT (CLASS_VALUE, PROPERTY_VALUE);
END_TYPE;

TYPE CLASS_VALUE= STRING;
END_TYPE;

TYPE PROPERTY_VALUE= STRING;
END_TYPE;

END_SCHEMA;
```

Appendix B

Preference Model with Express Language

UML Figures

Figure 1: Resource and Resource Instance with UML

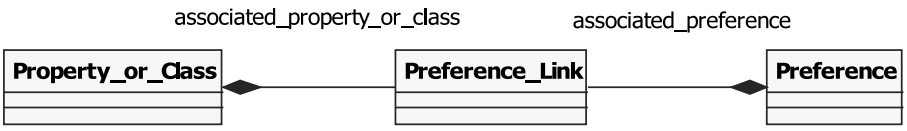


Figure 2: UML Representation of Preference Link Approach

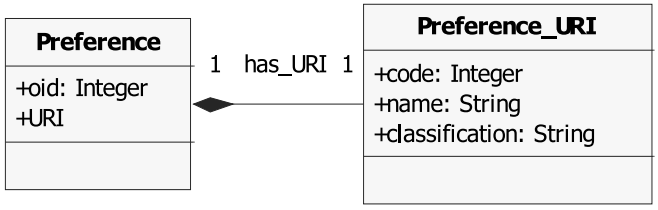


Figure 3: Graphical Representation of Preference Model.

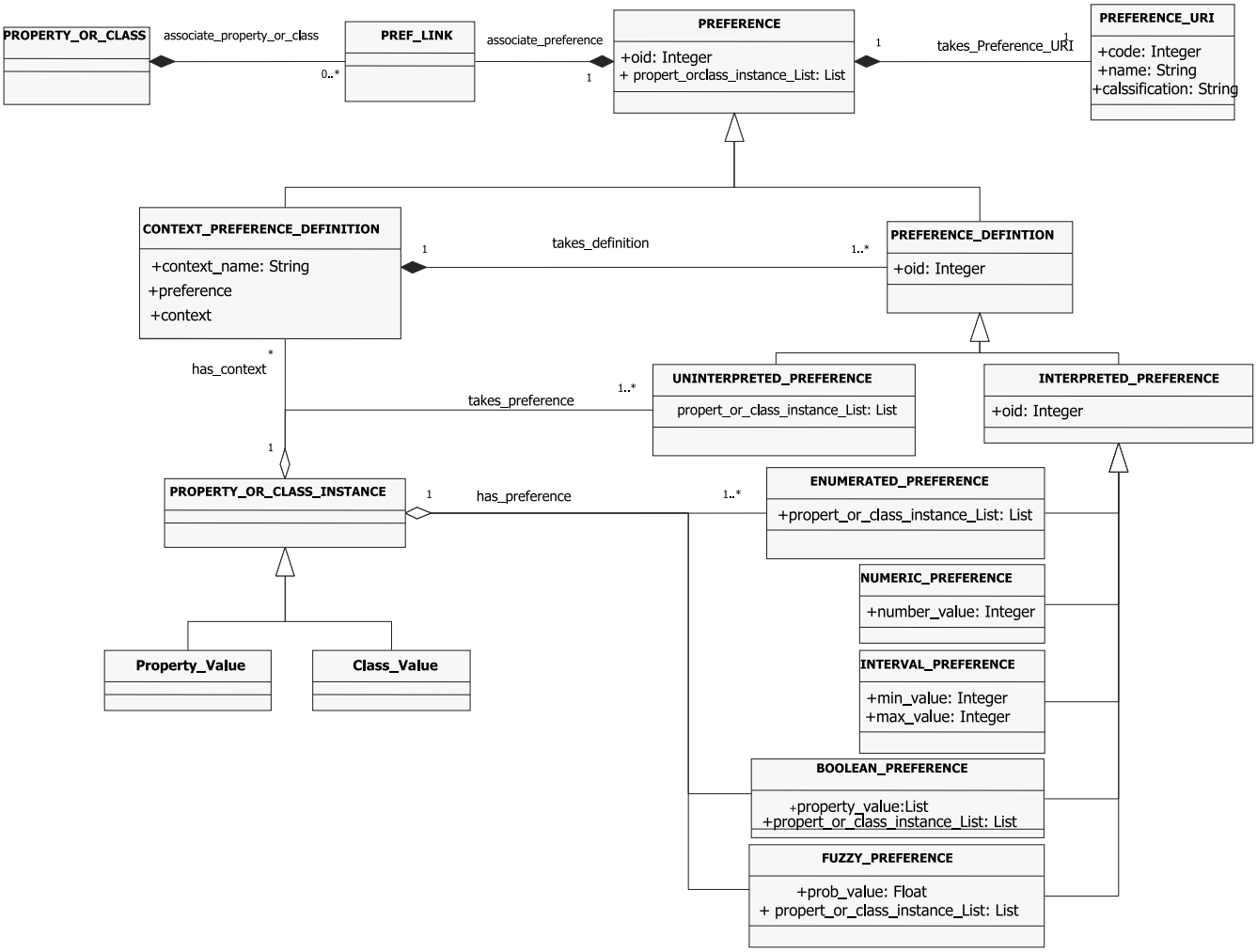


Figure 4: UML Representation of Preference Model

Appendix C

Complete syntax of the OntoQL language

This annex defines the grammatical rules of OntoQL language

Tokens

The following rules specify the tokens used for language definition, manipulation and querying OntoQL.

The keywords:

This sub-section gives the rules for the keywords of OntoQL language.

⟨*ABS*⟩ ::= abs
⟨*ADD*⟩ ::= add
⟨*ALL*⟩ ::= all
⟨*ALTER*⟩ ::= alter
⟨*AND*⟩ ::= and
⟨*ANY*⟩ ::= any
⟨*ARRAY*⟩ ::= array
⟨*AS*⟩ ::= as
⟨*ASC*⟩ ::= asc
⟨*ATTRIBUTE*⟩ ::= attribute
⟨*AVG*⟩ ::= avg
⟨*BETWEEN*⟩ ::= between
⟨*BOOLEAN*⟩ ::= boolean
⟨*CARDINALITY*⟩ ::= cardinality
⟨*CASE*⟩ ::= case
⟨*CAST*⟩ ::= cast
⟨*CHECK*⟩ :: check
⟨*COALESCE*⟩ ::= coalesce
⟨*COLUMN*⟩ ::= column
⟨*CONSTRAINT*⟩ ::= constraint
⟨*COUNT*⟩ ::= count
⟨*CREATE*⟩ ::= create
⟨*CROSS*⟩ ::= cross
⟨*DATE*⟩ ::= date
⟨*DELETE*⟩ ::= delete
⟨*DEREF*⟩ ::= deref
⟨*DERIVED*⟩ ::= derived
⟨*DESC*⟩ ::= desc
⟨*DESCRIPTOR*⟩ ::= descriptor
⟨*DISTINCT*⟩ ::= distinct
⟨*DROP*⟩ ::= drop
⟨*ELSE*⟩ ::= else

⟨*END*⟩ ::= end
⟨*ENTITY*⟩ ::= entity
⟨*ESCAPE*⟩ ::= escape
⟨*EXCEPT*⟩ ::= except
⟨*EXISTS*⟩ ::= exists
⟨*EXP*⟩ ::= exp
⟨*EXTENT*⟩ ::= extent
⟨*FALSE*⟩ ::= false
⟨*FLOAT*⟩ ::= float
⟨*FLOOR*⟩ ::= floor
⟨*FOR*⟩ ::= for
⟨*FOREIGN*⟩ ::= foreign
⟨*FROM*⟩ ::= from
⟨*FULL*⟩ ::= full
⟨*GROUP BY*⟩ ::= group by
⟨*HAVING*⟩ ::= having
⟨*IN*⟩ ::= in
⟨*INNER*⟩ ::= inner
⟨*INSERT*⟩ ::= insert
⟨*INT*⟩ ::= int
⟨*INTEGER*⟩ ::= integer
⟨*INTERSECT*⟩ ::= intersect
⟨*INTO*⟩ ::= into
⟨*IS*⟩ ::= is
⟨*JOIN*⟩ ::= join
⟨*KEY*⟩ ::= key
⟨*LANGUAGE*⟩ ::= language
⟨*LEFT*⟩ ::= left
⟨*LIKE*⟩ ::= like
⟨*LN*⟩ ::= ln
⟨*LOWER*⟩ ::= lower
⟨*MAX*⟩ ::= max
⟨*MIN*⟩ ::= min
⟨*MOD*⟩ ::= mod
⟨*MULTILINGUAL*⟩ ::= multilingual
⟨*NAMESPACE*⟩ ::= namespace
⟨*NATURAL*⟩ ::= natural
⟨*NONE*⟩ ::= none
⟨*NOT*⟩ ::= not
⟨*NULL*⟩ ::= null
⟨*NULLIF*⟩ ::= nullif
⟨*OF*⟩ ::= of
⟨*ON*⟩ ::= on
⟨*ONLY*⟩ ::= only
⟨*OR*⟩ ::= or
⟨*ORDER BY*⟩ ::= order by
⟨*OUTER*⟩ ::= outer
⟨*POWER*⟩ ::= power
⟨*PRIMARY*⟩ ::= primary
⟨*PROPERTY*⟩ ::= property
⟨*REAL*⟩ ::= real
⟨*REF*⟩ ::= ref
⟨*REFERENCES*⟩ ::= references
⟨*RIGHT*⟩ ::= right
⟨*SELECT*⟩ ::= select
⟨*SET*⟩ ::= set
⟨*SIMILAR*⟩ ::= similar
⟨*SOME*⟩ ::= some
⟨*SQRT*⟩ ::= sqrt
⟨*STRING*⟩ ::= string
⟨*SUBSTRING*⟩ ::= substring
⟨*SUM*⟩ ::= sum
⟨*TABLE*⟩ ::= table
⟨*THEN*⟩ ::= then
⟨*TREAT*⟩ ::= treat
⟨*TRUE*⟩ ::= true
⟨*TYPEOF*⟩ ::= typeof
⟨*UNDER*⟩ ::= under

⟨*UNION*⟩ ::= union
⟨*UNIQUE*⟩ ::= unique
⟨*UNNEST*⟩ ::= unnest
⟨*UPDATE*⟩ ::= update
⟨*UPPER*⟩ ::= upper
⟨*USING*⟩ ::= using
⟨*VALUES*⟩ ::= values
⟨*VARCHAR*⟩ ::= varchar
⟨*VIEW*⟩ ::= view
⟨*WHEN*⟩ ::= when
⟨*WHERE*⟩ ::= where

The lexical items

The following rules indicate how certain combinations of characters are interpreted as lexical items in the language. Numbers:

⟨*unsigned numeric literal*⟩ ::= ⟨*exact numeric literal*⟩
| ⟨*approximate numeric literal*⟩

⟨*exact numeric literal*⟩ ::= ⟨*unsigned integer*⟩ [. [⟨*unsigned integer*⟩]]
| . ⟨*unsigned integer*⟩

⟨*unsigned integer*⟩ ::= ⟨*digit*⟩ { ⟨*digit*⟩ }

⟨*digit*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

⟨*approximate numeric literal*⟩ ::= ⟨*exact numeric literal*⟩ E ⟨*signed integer*⟩

⟨*signed integer*⟩ ::= [⟨*sign*⟩] ⟨*unsigned integer*⟩

⟨*sign*⟩ ::= + | -

Strings, dates and booleans:

⟨*general literal*⟩ ::= ⟨*character string literal*⟩
| ⟨*date literal*⟩
| ⟨*boolean literal*⟩

⟨*character string literal*⟩ ::= ' [⟨*character representation list*⟩] '

⟨*character representation*⟩ ::= ⟨*nonquote character*⟩
| ⟨*quote symbol*⟩

⟨*nonquote character*⟩ ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

⟨*quote symbol*⟩ ::= ''

⟨*date literal*⟩ ::= DATE ' ⟨*date value*⟩ '

⟨*date value*⟩ ::= ⟨*unsigned integer*⟩ - ⟨*unsigned integer*⟩ - ⟨*unsigned integer*⟩

⟨*boolean literal*⟩ ::= TRUE | FALSE

The lexical item identifier used to reference the different elements manipulated by the language OntoQL:

⟨*identifier*⟩ ::= ⟨*identifier start*⟩ { ⟨*identifier part*⟩ }

⟨*identifier part*⟩ ::= ⟨*identifier start*⟩
| ⟨*identifier extend*⟩

⟨*identifier start*⟩ ::= _ | ⟨*nonquote character*⟩

⟨*identifier extend*⟩ ::= $ | ⟨*digit*⟩

identifiers

The following rules define the identifiers of the different elements manipulated by the language OntoQL

⟨*table name*⟩ ::= ⟨*identifier*⟩

⟨*column name*⟩ ::= ⟨*identifier*⟩

⟨*constraint name*⟩ ::= ⟨*identifier*⟩

⟨*alias name*⟩ ::= ⟨*identifier*⟩

⟨*function name*⟩ ::= ⟨*identifier*⟩

⟨*class id*⟩ ::= ⟨*identifier*⟩

⟨*property id*⟩ ::= ⟨*identifier*⟩

⟨*entity id*⟩ ::= # ⟨*identifier*⟩

⟨*attribute id*⟩ ::= # ⟨*identifier*⟩

⟨*category id*⟩ ::= ⟨*table name*⟩
| ⟨*class id*⟩
| ⟨*entity id*⟩

⟨*category id polymorph*⟩ ::= ⟨*table name*⟩
| ⟨*class id*⟩ | ONLY (⟨*class id*⟩)
| ⟨*entity id*⟩ | ONLY (⟨*entity id*⟩)

⟨*description id*⟩ ::= ⟨*column name*⟩
| ⟨*property id*⟩
| ⟨*attribute id*⟩

⟨*namespace id*⟩ ::= ⟨*identifier*⟩

⟨*namespace alias*⟩ ::= ⟨*identifier*⟩

⟨*language id*⟩ ::= AA | AB | AF | AM | AR | AS | AY | AZ | BA | BE | BG | BH | BI | BN | BO | BR | CA |
CO | CS | CY | DA | DE | DZ | EL | EN | EO | ES | ET | EU | FA | FI | FJ | FO | FR |
FY | GA | GD | GL | GN | GU | HA | HI | HR | HU | HY | IA | IE | IK | IN | IS | IT |
IW | JA | JI | JW | KA | KK | KL | KM | KN | KO | KS | KU | KY | LA | LN | LO | LT |
LV | MG | MI | MK | ML | MN | MO | MR | MS | MT | MY | NA | NE | NL | NO | OC | OM |
OR | PA | PL | PS | PT | QU | RM | RN | RO | RU | RW | SA | SD | SG | SH | SI | SK |
SL | SM | SN | SO | SQ | SR | SS | ST | SU | SV | SW | TA | TE | TG | TH | TI | TK |
TL | TN | TO | TR | TS | TT | TW | UK | UR | UZ | VI | VO | WO | XH | YO | ZH | ZU

Resources

In this section, we define the elements of grammar used by the various languages offered by the language OntoQL.

Data types

⟨*data type*⟩ ::= ⟨*predefined type*⟩
| ⟨*reference type*⟩
| ⟨*collection type*⟩

⟨*predefined type*⟩ ::= ⟨*character string type*⟩
| ⟨*numeric type*⟩
| ⟨*boolean type*⟩
| ⟨*date type*⟩

⟨*character string type*⟩ ::= [`MULTILINGUAL`] `STRING` [`(` ⟨*integer*⟩ `)`]
| [`MULTILINGUAL`] `VARCHAR` `(` ⟨*integer*⟩ `)`

⟨*numeric type*⟩ ::= ⟨*exact numeric type*⟩
| ⟨*approximate numeric type*⟩

⟨*exact numeric type*⟩ ::= `INT`
| `INTEGER`

⟨*approximate numeric type*⟩ ::= `FLOAT` [`(` ⟨*integer*⟩ `)`]
| `REAL`

⟨*boolean type*⟩ ::= `BOOLEAN`

⟨*date type*⟩ ::= `DATE`

⟨*reference type*⟩ ::= `REF` `(` ⟨*referenced type*⟩ `)`

⟨*referenced type*⟩ ::= ⟨*class id*⟩
| ⟨*entity id*⟩

⟨*collection type*⟩ ::= ⟨*data type*⟩ `ARRAY` [[⟨*integer*⟩]]

The values

The following rules define the syntax elements defining the values of types of data presented previously

⟨*value expression*⟩ ::= ⟨*numeric value expression*⟩
| ⟨*string value expression*⟩
| ⟨*collection value expression*⟩
| ⟨*boolean value expression*⟩

Integer values:

⟨*numeric value expression*⟩ ::= ⟨*term*⟩
| ⟨*numeric value expression*⟩ + ⟨*term*⟩
| ⟨*numeric value expression*⟩ - ⟨*term*⟩

⟨*term*⟩ ::= ⟨*factor*⟩
| ⟨*term*⟩ * ⟨*factor*⟩
| ⟨*term*⟩ / ⟨*factor*⟩

⟨*factor*⟩ ::= [-] ⟨*numeric primary*⟩

⟨*numeric primary*⟩ ::= ⟨*unsigned numeric literal*⟩
| ⟨*value expression primary*⟩
| ⟨*numeric value function*⟩

⟨*numeric value function*⟩ ::= ⟨*cardinality expression*⟩
| ⟨*absolute value expression*⟩
| ⟨*modulus expression*⟩
| ⟨*natural logarithm*⟩
| ⟨*exponential function*⟩
| ⟨*power function*⟩
| ⟨*square root*⟩
| ⟨*floor function*⟩

⟨*cardinality expression*⟩ ::= CARDINALITY (⟨*collection value expression*⟩)

⟨*absolute value expression*⟩ ::= ABS (⟨*numeric value expression*⟩)

⟨*modulus expression*⟩ ::= MOD (⟨*numeric value expression*⟩ , ⟨*numeric value expression*⟩)

⟨*natural logarithm*⟩ ::= LN (⟨*numeric value expression*⟩)

⟨*exponential function*⟩ ::= EXP (⟨*numeric value expression*⟩)

⟨*power function*⟩ ::= POWER (⟨*numeric value expression*⟩ ,
⟨*numeric value expression*⟩)

⟨*square root*⟩ ::= SQRT (⟨*numeric value expression*⟩)

⟨*floor function*⟩ ::= FLOOR (⟨*numeric value expression*⟩)

Values of type string:

⟨*string value expression*⟩ ::= ⟨*concatenation*⟩ | ⟨*character factor*⟩

⟨*concatenation*⟩ ::= ⟨*string value expression*⟩ || ⟨*character factor*⟩

⟨*character factor*⟩ ::= ⟨*character primary*⟩

⟨*character primary*⟩ ::= ⟨*character string literal*⟩
| ⟨*value expression primary*⟩
| ⟨*string value function*⟩

⟨*string value function*⟩ ::= ⟨*character substring function*⟩
| ⟨*regexpr substring function*⟩
| ⟨*fold*⟩

⟨*character substring function*⟩ ::= SUBSTRING (⟨*string value expression*⟩
FROM ⟨*numeric value expression*⟩
[FOR ⟨*numeric value expression*⟩])

⟨*regexpr substring function*⟩ ::= `SUBSTRING` `(` ⟨*string value expression*⟩
`SIMILAR` ⟨*string value expression*⟩
`ESCAPE` ⟨*string value expression*⟩ `)`

⟨*fold*⟩ ::= ⟨*fold op*⟩ `(` ⟨*string value expression*⟩ `)`

⟨*fold op*⟩ ::= `UPPER` | `LOWER`

Boolean values:

⟨*boolean value expression*⟩ ::= ⟨*boolean term*⟩
| ⟨*boolean value expression*⟩ `OR` ⟨*boolean term*⟩

⟨*boolean term*⟩ ::= ⟨*boolean factor*⟩
| ⟨*boolean term*⟩ `AND` ⟨*boolean factor*⟩

⟨*boolean factor*⟩ ::= [`NOT`] ⟨*boolean test*⟩

⟨*boolean test*⟩ ::= ⟨*predicate*⟩ | ⟨*boolean predicand*⟩

⟨*predicate*⟩ ::= ⟨*comparison predicate*⟩
| ⟨*between predicate*⟩
| ⟨*in predicate*⟩
| ⟨*like predicate*⟩
| ⟨*null predicate*⟩
| ⟨*quantified predicate*⟩
| ⟨*exists predicate*⟩
| ⟨*type predicate*⟩

⟨*comparison predicate*⟩ ::= ⟨*value expression*⟩ ⟨*equality op*⟩ ⟨*value expression*⟩

⟨*equality op*⟩ ::= = | <> | > | >= | < | <=

⟨*between predicate*⟩ ::= ⟨*value expression*⟩ [`NOT`] `BETWEEN`
⟨*value expression*⟩ `AND` ⟨*value expression*⟩

⟨*in predicate*⟩ ::= [`NOT`] `IN` ⟨*in predicate value*⟩

⟨*in predicate value*⟩ ::= ⟨*subquery*⟩ | `(` ⟨*value expression list*⟩ `)`

⟨*like predicate*⟩ ::= ⟨*value expression*⟩ [`NOT`] `LIKE` ⟨*value expression*⟩

⟨*null predicate*⟩ ::= ⟨*value expression*⟩ `IS` [`NOT`] `NULL`

⟨*quantified predicate*⟩ ::= ⟨*value expression*⟩ ⟨*quantifier op*⟩ `(` ⟨*sub query*⟩ `)`

⟨*quantifier op*⟩ ::= `ALL` | `SOME` | `ANY`

⟨*exists predicate*⟩ ::= `EXISTS` ⟨*subquery*⟩

⟨*type predicate*⟩ ::= ⟨*value expression*⟩ `IS` [`NOT`] `OF` `(` ⟨*type list*⟩ `)`

⟨*type list*⟩ ::= [`ONLY`] ⟨*is of type*⟩ { `,` [`ONLY`] ⟨*is of type*⟩ }

⟨*is of type*⟩ ::= ⟨*reference type*⟩ | ⟨*class id*⟩ | ⟨*entity id*⟩

⟨*boolean predicand*⟩ ::= `(` ⟨*boolean value expression*⟩ `)`
| ⟨*boolean literal*⟩
| ⟨*value expression primary*⟩

Values of type collection:

⟨*collection value expression*⟩ ::= ⟨*array concatenation*⟩
| ⟨*array primary*⟩

⟨*array concatenation*⟩ ::= ⟨*collection value expression*⟩ `||` ⟨*array primary*⟩

⟨*array primary*⟩ ::= ⟨*value expression primary*⟩
| ⟨*array value constructor*⟩

⟨*array value constructor*⟩ ::= ⟨*array value constructor by enumeration*⟩
| ⟨*array value constructor by query*⟩

⟨*array value constructor by enumeration*⟩ ::= `ARRAY [` ⟨*value expression list*⟩ `]`

⟨*array value constructor by query*⟩ ::= `ARRAY (` ⟨*query expression*⟩ `)`

Expressions producing different types of values:

⟨*value expression primary*⟩ ::= ⟨*par value expression*⟩
| ⟨*nonpar value expression primary*⟩

⟨*par value expression*⟩ ::= `(` ⟨*value expression*⟩ `)`

⟨*nonpar value expression primary*⟩ ::= ⟨*description reference*⟩
| ⟨*scalar subquery*⟩
| ⟨*function call*⟩
| ⟨*aggregate function*⟩
| ⟨*case expression*⟩
| ⟨*cast specification*⟩
| ⟨*subtype treatment*⟩
| ⟨*typeof treatment*⟩
| ⟨*reference resolution*⟩
| ⟨*null specification*⟩

⟨*description reference*⟩ ::= [⟨*identifier*⟩ `.`] ⟨*qualified description*⟩

⟨*qualified description*⟩ ::= ⟨*column name*⟩
| ⟨*property path expression*⟩
| ⟨*attribute path expression*⟩
| ⟨*identifier*⟩

⟨*property path expression*⟩ ::= ⟨*property id*⟩ { `.` ⟨*property id*⟩ }

⟨*attribute path expression*⟩ ::= ⟨*attribute id*⟩ { `.` ⟨*attribute id*⟩ }

⟨*scalar subquery*⟩ ::= ⟨*subquery*⟩

⟨*function call*⟩ ::= ⟨*function name*⟩ `(` [⟨*value expression list*⟩] `)`

⟨*aggregate function*⟩ ::= `COUNT ( * )` | ⟨*general set function*⟩

⟨*general set function*⟩ ::= ⟨*computational operation*⟩
`(` [⟨*set quantifier*⟩] ⟨*value expression*⟩ `)`

⟨*computational operation*⟩ ::= AVG | MAX | MIN | SUM | COUNT

⟨*set quantifier*⟩ ::= DISTINCT | ALL

⟨*case expression*⟩ ::= ⟨*case abbreviation*⟩
| ⟨*case specification*⟩

⟨*case abbreviation*⟩ ::= NULLIF (⟨*value expression*⟩ , ⟨*value expression*⟩)
| COALESCE (⟨*value expression list*⟩)

⟨*case specification*⟩ ::= ⟨*simple case*⟩
| ⟨*searched case*⟩

⟨*simple case*⟩ ::= CASE ⟨*value expression*⟩ ⟨*simple when clause list*⟩
[⟨*else clause*⟩] END

⟨*simple when clause*⟩ ::= WHEN ⟨*value expression*⟩ THEN ⟨*value expression*⟩

⟨*else clause*⟩ ::= ELSE ⟨*value expression*⟩

⟨*searched case*⟩ ::= CASE ⟨*searched when clause list*⟩ [⟨*else clause*⟩] END

⟨*searched when clause*⟩ ::= WHEN ⟨*search condition*⟩ THEN ⟨*value expression*⟩

⟨*search condition*⟩ ::= ⟨*boolean value expression*⟩

⟨*cast specification*⟩ ::= CAST (⟨*value expression*⟩ AS ⟨*data type*⟩)

⟨*subtype treatment*⟩ ::= TREAT (⟨*value expression*⟩ AS ⟨*target subtype*⟩)

⟨*target subtype*⟩ ::= ⟨*reference type*⟩ | ⟨*class id*⟩ | ⟨*entity id*⟩

⟨*typeof treatment*⟩ ::= TYPEOF (⟨*value expression*⟩)

⟨*reference resolution*⟩ ::= DEREF (⟨*value expression*⟩)

⟨*null specification*⟩ ::= NULL

Data def nition language DDL

The following rules define the syntax of the language data definition language OntoQL. The DDL level logic:

⟨*table definition*⟩ ::= CREATE TABLE ⟨*table name*⟩ ⟨*table element list*⟩

⟨*table element*⟩ ::= ⟨*column definition*⟩
| ⟨*table constraint definition*⟩

⟨*column definition*⟩ ::= ⟨*column name*⟩ ⟨*data type*⟩
{ ⟨*column constraint definition*⟩ }

⟨*column constraint definition*⟩ ::= NOT NULL
| ⟨*unique specification*⟩
| ⟨*references specification*⟩
| ⟨*check constraint definition*⟩

⟨*unique specification*⟩ ::= UNIQUE | PRIMARY KEY

⟨*references specification*⟩ ::= REFERENCES ⟨*table name*⟩ [(⟨*column name list*⟩)]

⟨*check constraint definition*⟩ ::= `CHECK` `(` ⟨*search condition*⟩ `)`

⟨*table constraint definition*⟩ ::= [⟨*constraint name definition*⟩] ⟨*table constraint*⟩

⟨*constraint name definition*⟩ ::= `CONSTRAINT` ⟨*constraint name*⟩

⟨*table constraint*⟩ ::= ⟨*unique constraint definition*⟩
| ⟨*referential constraint definition*⟩
| ⟨*check constraint definition*⟩

⟨*unique constraint definition*⟩ ::= ⟨*unique specification*⟩ `(` ⟨*column name list*⟩ `)`

⟨*referential constraint definition*⟩ ::= `FOREIGN KEY` `(` ⟨*column name list*⟩ `)` ⟨*references specification*⟩

⟨*alter table statement*⟩ ::= `ALTER TABLE` ⟨*table name*⟩ ⟨*alter table action*⟩

⟨*alter table action*⟩ ::= ⟨*add column definition*⟩
| ⟨*drop column definition*⟩
| ⟨*add table constraint definition*⟩
| ⟨*drop table constraint definition*⟩

⟨*add column definition*⟩ ::= `ADD` ⟨*column definition*⟩

⟨*drop column definition*⟩ ::= `DROP` ⟨*column name*⟩

⟨*add table constraint definition*⟩ ::= `ADD` ⟨*table constraint definition*⟩

⟨*drop table constraint definition*⟩ ::= `DROP CONSTRAINT` ⟨*constraint name*⟩

⟨*drop table statement*⟩ ::= `DROP TABLE` ⟨*table name*⟩

DDL at the ontological level:

⟨*class definition*⟩ ::= `CREATE` ⟨*entity id*⟩ ⟨*class id*⟩ [⟨*view clause*⟩] [⟨*under clause*⟩]
[⟨*descriptor clause*⟩] [⟨*properties clause list*⟩]

⟨*view clause*⟩ ::= `AS VIEW`

⟨*under clause*⟩ ::= `UNDER` ⟨*class id list*⟩

⟨*descriptor clause*⟩ ::= `DESCRIPTOR` `(` ⟨*attribute value list*⟩ `)`

⟨*attribute value*⟩ ::= ⟨*attribute id*⟩ = ⟨*value expression*⟩

⟨*properties clause*⟩ ::= ⟨*entity id*⟩ `(` ⟨*property definition list*⟩ `)`

⟨*property definition*⟩ ::= ⟨*prop id*⟩ ⟨*datatype*⟩ [⟨*descriptor clause*⟩]

⟨*alter class statement*⟩ ::= `ALTER` ⟨*class id*⟩ [⟨*descriptor clause*⟩]
[⟨*alter class action*⟩]

⟨*alter class action*⟩ ::= ⟨*add property definition*⟩
| ⟨*drop property definition*⟩

⟨*add property definition*⟩ ::= `ADD` [⟨*entity id*⟩] ⟨*property definition*⟩ [⟨*descriptor clause*⟩]

⟨*drop property definition*⟩ ::= `DROP` ⟨*property id*⟩

⟨*drop class definition*⟩ ::= `DROP` ⟨*class id*⟩

It also helps to define the extensions of classes:

⟨*extension definition*⟩ ::= `CREATE EXTENT OF` ⟨*class id*⟩ `(` ⟨*property id list*⟩ `)` [⟨*logical clause*⟩]

⟨*logical clause*⟩ ::= `TABLE` [⟨*table and column name*⟩]

⟨*table and column name*⟩ ::= ⟨*table name*⟩ [`(` ⟨*column name list*⟩ `)`]

⟨*alter extension statement*⟩ ::= `ALTER EXTENT OF` ⟨*class id*⟩ ⟨*alter extent action*⟩

⟨*alter extension action*⟩ ::= ⟨*add property definition*⟩
| ⟨*drop property definition*⟩

⟨*add property definition*⟩ ::= `ADD` [`PROPERTY`] ⟨*property id*⟩ [`COLUMN` ⟨*column name*⟩]

⟨*drop property definition*⟩ ::= `DROP` [`PROPERTY`] ⟨*property id*⟩

⟨*drop extension statement*⟩ ::= `DROP EXTENT OF` ⟨*class id*⟩

⟨*entity definition*⟩ ::= `CREATE ENTITY` ⟨*entity id*⟩ [⟨*under clause*⟩] ⟨*attribute clause*⟩

⟨*under clause*⟩ ::= `UNDER` ⟨*entity id list*⟩

⟨*attribute clause*⟩ ::= ⟨*attribute definition list*⟩

⟨*attribute definition*⟩ ::= ⟨*attribute id*⟩ ⟨*datatype*⟩ [⟨*derived clause*⟩]

⟨*derived clause*⟩ ::= `DERIVED BY` ⟨*function name*⟩

⟨*alter entity statement*⟩ ::= `ALTER ENTITY` ⟨*entity id*⟩ ⟨*alter entity action*⟩

⟨*alter entity action*⟩ ::= ⟨*add attribute definition*⟩
| ⟨*drop attribute definition*⟩

⟨*add attribute definition*⟩ ::= `ADD` [`ATTRIBUTE`] ⟨*attribute definition*⟩

⟨*drop attribute definition*⟩ ::= `DROP` [`ATTRIBUTE`] ⟨*attribute id*⟩

⟨*drop entity statement*⟩ ::= `DROP ENTITY` ⟨*entity id*⟩

Language for manipulating data: DML

The following rules define the syntax of the language data manipulation language OntoQL:

⟨*insert statement*⟩ ::= `INSERT INTO` ⟨*category id*⟩ ⟨*insert description and source*⟩

⟨*insert description and source*⟩ ::= ⟨*from subquery*⟩ | ⟨*from constructor*⟩

⟨*from subquery*⟩ ::= [`(` ⟨*insert description list*⟩ `)`] ⟨*query expression*⟩

⟨*insert description list*⟩ ::= ⟨*column name list*⟩
| ⟨*property id list*⟩
| ⟨*attribute id list*⟩

⟨*from constructor*⟩ ::= [`(` ⟨*insert description list*⟩ `)`] ⟨*values clause*⟩

⟨*values clause*⟩ ::= `VALUES (` ⟨*values expression list*⟩ `)`

⟨*update statement*⟩ ::= `UPDATE` ⟨*category id polymorph*⟩ `SET` ⟨*set clause list*⟩
[`WHERE` ⟨*search condition*⟩]

⟨*set clause*⟩ ::= ⟨*description id*⟩ = ⟨*value expression*⟩

⟨*delete statement*⟩ ::= `DELETE FROM` ⟨*category id polymorph*⟩
[`WHERE` ⟨*search condition*⟩]

Query language of data

The following rules define the syntax of the query language language OntoQL:

⟨*query expression*⟩ ::= ⟨*query term*⟩
| ⟨*query expression*⟩ `UNION` ⟨*set quantifier*⟩ ⟨*query term*⟩
| ⟨*query expression*⟩ `EXCEPT` ⟨*set quantifier*⟩ ⟨*query term*⟩

⟨*query term*⟩ ::= ⟨*query primary*⟩
| ⟨*query term*⟩ `INTERSECT` ⟨*set quantifier*⟩ ⟨*query primary*⟩

⟨*query primary*⟩ ::= ⟨*query specification*⟩ | `(` ⟨*query expression*⟩ `)`

⟨*query specification*⟩ ::= ⟨*select clause*⟩ ⟨*from clause*⟩ [⟨*where clause*⟩]
[⟨*group by clause*⟩] [⟨*having clause*⟩] [⟨*order by clause*⟩]
[⟨*namespace clause*⟩] [⟨*language clause*⟩]

`SELECT` Clause:

⟨*select clause*⟩ ::= `SELECT` [⟨*set quantifier*⟩] ⟨*select list*⟩

⟨*select list*⟩ ::= * | ⟨*select sublist*⟩ { `,` ⟨*select sublist*⟩ }

⟨*select sublist*⟩ ::= ⟨*value expression*⟩ [⟨*as clause*⟩]

⟨*as clause*⟩ ::= [`AS`] ⟨*alias name*⟩

`FROM` clause:

⟨*from clause*⟩ ::= `FROM` ⟨*category reference list*⟩

⟨*category reference*⟩ ::= ⟨*category primary*⟩
| ⟨*joined category*⟩

⟨*category primary*⟩ ::= ⟨*category or subquery*⟩ [[`AS`] ⟨*alias name*⟩]
| ⟨*collection derived category*⟩ [`AS`] ⟨*alias name*⟩

⟨*category or subquery*⟩ ::= ⟨*category id polymorph*⟩
| ⟨*dynamic iterator*⟩
| ⟨*subquery*⟩

⟨*dynamic iterator*⟩ ::= ⟨*identifier*⟩ | `ONLY (` ⟨*identifier*⟩ `)`

⟨*subquery*⟩ ::= `(` ⟨*query expression*⟩ `)`

⟨*collection derived category*⟩ ::= `UNNEST (` ⟨*collection value expression*⟩ `)`

⟨*joined category*⟩ ::= ⟨*cross join*⟩
| ⟨*qualified join*⟩
| ⟨*natural join*⟩

⟨*cross join*⟩ ::= ⟨*category reference*⟩ `CROSS JOIN` ⟨*category primary*⟩

⟨*qualified join*⟩ ::= ⟨*category reference*⟩[⟨*join type*⟩] `JOIN`
⟨*categroy reference*⟩ ⟨*join specification*⟩

⟨*join type*⟩ ::= `INNER` | ⟨*outer join type*⟩ [`OUTER`]

⟨*outer join type*⟩ ::= `LEFT` | `RIGHT` | `FULL`

⟨*join specification*⟩ ::= ⟨*join condition*⟩
| ⟨*named columns join*⟩

⟨*join condition*⟩ ::= `ON` ⟨*search condition*⟩

⟨*named columns join*⟩ ::= `USING (` ⟨*description id list*⟩ `)`

⟨*natural join*⟩ ::= ⟨*category reference*⟩ `NATURAL` [⟨*join type*⟩]
`JOIN` ⟨*category primary*⟩

`WHERE`, `GROUP BY`, `HAVING` and `ORDER BY` Clauses:

⟨*where clause*⟩ ::= `WHERE` ⟨*search condition*⟩

⟨*group by clause*⟩ ::= `GROUP BY` [⟨*set quantifier*⟩] ⟨*description id list*⟩

⟨*having clause*⟩ ::= `HAVING` ⟨*search condition*⟩

⟨*order by*⟩ ::= `ORDER BY` ⟨*sort specification list*⟩

⟨*sort specification*⟩ ::= ⟨*sort key*⟩ [⟨*ordering specification*⟩]

⟨*sort key*⟩ ::= ⟨*value expression*⟩

⟨*ordering specification*⟩ ::= `ASC` | `DESC`

`NAMESPACE` and `LANGUAGE` Clauses:

⟨*preffering clause*⟩ ::= `PREFERRING` ⟨*search condition*⟩

⟨*group by clause*⟩ ::= `GROUP BY` [⟨*set quantifier*⟩] ⟨*description id list*⟩

⟨*having clause*⟩ ::= `HAVING` ⟨*search condition*⟩

⟨*order by*⟩ ::= `ORDER BY` ⟨*sort specification list*⟩

⟨*sort specification*⟩ ::= ⟨*sort key*⟩ [⟨*ordering specification*⟩]

⟨*sort key*⟩ ::= ⟨*value expression*⟩

⟨*ordering specification*⟩ ::= `ASC` | `DESC`

`NAMESPACE`and `LANGUAGE` Clauses:

⟨*namespace clause*⟩ ::= `USING NAMESPACE` ⟨*namespace definition list*⟩

⟨*namespace definition*⟩ ::= [⟨*namespace alias*⟩ =] ⟨*namespace id*⟩

⟨*language clause*⟩ ::= `USING LANGUAGE` ⟨*language id*⟩

Data View Language : DVL

The following rules define the syntax for defining views with language OntoQL.

⟨*view definition*⟩ ::= `CREATE VIEW` ⟨*table name*⟩ ⟨*view specification*⟩
`AS` ⟨*query expression*⟩

⟨*view specification*⟩ ::= ⟨*regular view specification*⟩
| ⟨*referenceable view specification*⟩

⟨*regular view specification*⟩ ::= [`(` ⟨*column name list*⟩ `)`]

⟨*referenceable view specification*⟩ ::= `OF` ⟨*class id*⟩ [⟨*property id list*⟩]

Setting the language The language OntoQL is set by the namespace in which it is to find the elements of a ontology and the natural language in which it must recognize the names of the different elements handled. The syntax for setting the language OntoQL is as follows: rechercher:

⟨*global namespace definition*⟩ ::= `SET NAMESPACE` [⟨*namespace alias*⟩ =] ⟨*namespace specification*⟩

⟨*namespace specification*⟩ ::= ⟨*namespace id*⟩ | `NONE`

⟨*global language definition*⟩ ::= `SET LANGUAGE` ⟨*language specification*⟩

⟨*language specification*⟩ ::= ⟨*language id*⟩ | `NONE`

List of Figures

List of Tables

Abstract

De nos jours, les systèmes d'information gèrent de volumineuses données. Avec l'avènement du Web Sémantique, la quantité de données ontologiques (ou instances) disponibles s'est accrue. Permettre un accès personnalisé à ces données est devenue cruciale. Les utilisateurs sont submergés par les nombreux résultats fournis en réponse à leurs requêtes. Pour être utilisable, ces résultats doivent être filtrées et ordonnées. La capture et l'exploitation des préférences utilisateurs ont été proposées comme une solution à ce problème. Cependant, les approches existantes définissent habituellement les préférences pour une application donnée. Il est ainsi difficile de partager et réutiliser dans d'autres contextes les préférences capturées. Nous proposons une approche basée sur plusieurs modèles proposés au sein des communautés Bases de Données et Web Sémantique. Elle définit un model partageable et générique pour représenter les préférences utilisateurs, et incorpore plusieurs types de préférences de la littérature qui sont traités de manière séparée. L'idée sous-jacente à notre approche est de traiter les préférences de manière modulaire en les liant aux ontologies qui décrivent la sémantique des données gérées par les applications. Ainsi leur prise en compte se fait au niveau ontologique et non au niveau logique des données. La nouveauté de l'approche est que les préférences définies sont attachées aux ontologies, qui décrivent la sémantique des données manipulées par les applications. Le modèle de préférence est formellement défini en utilisant le langage de modélisation des données `EXPRESS` de manière à éviter toute ambiguïté du modèle. Par ailleurs, le modèle proposé offre un mécanisme de persistance et un langage d'interrogation dédié. Il est implémenté en utilisant un système de Bases de Données à Base Ontologique (BDBO) qui permet de gérer à la fois les ontologies et les données instances. Ceci permet d'offrir une description sémantique des préférences. Nous avons étendu le modèle des BDBO afin de supporter la prise en compte des préférences. L'implémentation a été faite dans le cadre de la BDBO OntoDB pour laquelle nous avons étendu le langage d'interrogation associé OntoQL. L'approche est illustrée à travers un cas d'étude dans le domaine du tourisme.

Keywords : Bases de Données à Base Ontologique, Personnalisation, Web Sémantique, Ontologie, Préférences Utilisateurs.

Abstract

Nowadays information systems manage huge amount of data. With the emergence of the Semantic Web, the amount of available ontological data (or instances) has increased. To allow personalized access to this information has become a crucial necessity. Users are overwhelmed by the numerous results provided in response to their requests. In order to be usable, these results must often be sorted and filtered. The capture and exploitation of user preferences have been proposed as a solution to this problem. However, the existing approaches usually define preferences for a particular application. Thus, it is difficult to share and reuse the handled preferences in other contexts. Our approach, which defines a sharable and generic model to represent user preferences, based on several models proposed in the Databases and the Semantic Web communities. It incorporates several types of preferences proposed in the literature, but are treated separately. Our idea is to address preferences of a modular way by linking them to ontologies for describing the semantics of the data, handled by the applications. It is thus to raise the treatment preferences of logic level (structure) to the ontological level. The novelty of our approach is that the defined preferences are attached to the ontologies, which describe the semantic of the data manipulated by the applications. The preference model is formally defined using the EXPRESS data modeling language, which ensures a free ambiguity definition. Moreover, the proposed model offers a persistence mechanism and a dedicated language; which is implemented using Ontology Based Databases (OBDB) system, that manages both ontologies and extended data instances, in order to support a semantic description of preferences. These databases are associated with explanation languages, supporting description, querying, etc. on both ontologies and data. Usually queries return a big amount of data that may be sorted in order to find the relevant ones. Moreover, in the current situation few approaches are considering user preferences, when querying has been developed. Yet this problem is fundamental for many applications, especially in the e-commerce domain. Our second approach, which defines preferences in terms of ontologies that describe the semantics of handled data, provides a mechanism for querying with preferences. Thus, an extension to existing ontology based query languages is proposed, for querying ontological data with preferences. The proposed extension has been implemented onto the OntoDB OBDB associated to the OntoQL query language and the approach is illustrated through a case study in the tourism domain.

Keywords : Ontology-based Database (OBDB), Personalization, Semantic Web, Ontology, User preferences.

A generic model for handling preferences in ontology based databases

Presented by:

Dilek TAPUCU

Thesis Advisors :

Yamine AIT-AMEUR and Murat Osman UNALIR

Abstract. Nowadays information systems manage huge amount of data. With the emergence of the Semantic Web, the amount of available ontological data (or instances) has increased. To allow personalized access to this information has become a crucial necessity. Users are overwhelmed by the numerous results provided in response to their requests. In order to be usable, these results must often be sorted and filtered. The capture and exploitation of user preferences have been proposed as a solution to this problem. However, the existing approaches usually define preferences for a particular application. Thus, it is difficult to share and reuse the handled preferences in other contexts. Our approach, which defines a sharable and generic model to represent user preferences, based on several models proposed in the Databases and the Semantic Web communities. It incorporates several types of preferences proposed in the literature, but are treated separately. Our idea is to address preferences of a modular way by linking them to ontologies for describing the semantics of the data, handled by the applications. It is thus to raise the treatment preferences of logic level (structure) to the ontological level. The novelty of our approach is that the defined preferences are attached to the ontologies, which describe the semantic of the data manipulated by the applications. The preference model is formally defined using the EXPRESS data modeling language, which ensures a free ambiguity definition. Moreover, the proposed model offers a persistence mechanism and a dedicated language; which is implemented using Ontology Based Databases (OBDB) system, that manages both ontologies and extended data instances, in order to support a semantic description of preferences. These databases are associated with explanation languages, supporting description, querying, etc. on both ontologies and data. Usually queries return a big amount of data that may be sorted in order to find the relevant ones. Moreover, in the current situation few approaches are considering user preferences, when querying has been developed. Yet this problem is fundamental for many applications, especially in the e-commerce domain. Our second approach, which defines preferences in terms of ontologies that describe the semantics of handled data, provides a mechanism for querying with preferences. Thus, an extension to existing ontology based query languages is proposed, for querying ontological data with preferences. The proposed extension has been implemented onto the OntoDB OBDB associated to the OntoQL query language and the approach is illustrated through a case study in the tourism domain.

Keywords: Ontology-based Database (OBDB), Personalization, Semantic Web, Ontology, User preferences.