SEMISTRUCTURED PROBABILISTIC OBJECT QUERY LANGUAGE (A Query Language for Semistructured Probabilistic Data)

Praveen Gutti
University of Kentucky, PraveenGutti@gmail.com

ABSTRACT OF THESIS

SEMISTRUCTURED PROBABILISTIC OBJECT QUERY LANGUAGE
(A Query Language for Semistructured Probabilistic Data)

This work presents SPOQL, a structured query language for Semistructured Probabilistic Object (SPO) model [4]. The original query language for semistructured probabilistic database management system [20], SP-Algebra [4], has limitations such as complex functional notation and unfamiliarity to application programmers. SPOQL alleviates these problems by providing a user friendly and familiar SQL-like declarative syntax for writing queries against SPDBMS. We show that parsing SPOQL queries is a more involving task than parsing SQL queries. We describe the evaluation algorithm for SPOQL queries that we have implemented.

KEYWORDS: Probability distribution, Semistructured Probabilistic Object (SPO), SP-Algebra, SPOQL, SPOQL semantics

Praveen Gutti

Summer 2007

SEMISTRUCTURED PROBABILISTIC OBJECT QUERY LANGUAGE

(A Query Language for Semistructured Probabilistic Data)

By

Praveen Gutti

Dr. Alexander Dekhtyar

Director of Thesis

Dr. Judy Goldsmith

Co-Director of Thesis

Dr. Raphael A. Finkel

Director of Graduate Studies

THESIS

Praveen Gutti

The Graduate School

University of Kentucky

2007.

SEMISTRUCTURED PROBABILISTIC OBJECT QUERY LANGUAGE

(A Query Language for Semistructured Probabilistic Data)

THESIS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the College of Engineering at the University of Kentucky

By

Praveen Gutti

Director: Dr. Alexander Dekhtyar, Assistant Professor of Computer Science

Co-Director: Dr. Judy Goldsmith, Professor of Computer Science

Lexington, Kentucky

2007.

Acknowledgments

I would like to express my gratitude to my advisor Dr. Alexander Dekhtyar, whose constant support and encouragement made this work possible. He was always supportive of my efforts and was a source of inspiration. I am grateful to him for teaching me what research is all about and for teaching me how to organize and convey my ideas effectively. His able guidance and expertise helped immensely in shaping up this work. I would like to express my gratefulness to my co-advisor Dr. Judy Goldsmith for the considerable amount of time she spent with me despite her busy schedule. Her valuable suggestions were very helpful in documenting this work.

I would also like to thank Krol Kevin Mathias, PhD. student at University of Kentucky, without whose contributions, this work would not have been possible. He was always available for discussions and his valuable suggestions were very helpful to this work. I would like to thank Dr. Wenzhong Zhao for introducing me to the semistructured probabilistic database management system he developed. I thank Dr. Kenneth L Calvert and Dr. D. Manivannan for taking their time to serve on my committee.

I would like to thank my family for encouraging me to take up higher education. Their support and blessings made everything possible for me. Finally, I thank God for showering his blessings up on me.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many vital applications, used in every walk of life, such as stock market prediction software, weather forecast software, image recognition and analysis software and other applications of Bayesian Nets [15] are built upon complex databases storing uncertain information. Relational databases do not provide consistent support for storing and querying probabilistic information. In order to model this uncertain information and provide support for storing and querying probabilistic information, researchers have proposed several relational models [12, 2, 6, 1], object oriented data models [11, 8] and semistructured models [10, 14, 4] over the last two decades. But none of the approaches proved flexible enough to handle probability distributions in different contexts. For example, consider stock market analysis, where the level of possible financial gain while buying the share of a company can be represented in many forms, such as simple probability distribution or joint probability distribution, depending upon the number of aspects selected such as the past financial history of a company.

With varying information formats, any of the current probabilistic models require separate storage, making it hard to express even simple queries. For example, to find all the probability distributions involving the aspect of company establishment date, the application has to query all relations having establishment date as one of their fields, resulting in multiple queries. In order to alleviate the above problem, Dekhtyar et al. proposed the semi-structured probabilistic object (SPO) data model [4,20], which provides support for storing and managing diverse probability distributions of discrete random variables with finite domains and associated information. That means, unlike the other data models, SPO data model can store and query probabilistic information with different formats.

The SPO data model is a semi-structured data model [20]. Semistructured data models are aimed at developing the database management techniques to store and retrieve uncertain data like probability distributions. Because of the similarity of semistructured data model and data models for the Extensible Markup language (XML), the widely accepted open standard for data storage and transmission over the internet, XML is used to represent the SPO objects in this SPO data model. This entire framework for storing and querying varying probabilistic information was implemented by Dr. Zhao [20] as Semistructured Probabilistic Database Management System(SPDBMS) with a semi structured probabilistic query algebra(SP-algebra) for manipulating and querying SPO objects.

SPDBMS lacked a high level query language for querying and retrieving SPO-objects. Even though SP-Algebra is well-defined and structured, it is quite terse to express queries with complex notations. It is more difficult for application programmers to express complex queries using SP-Algebra than with a language which has the feel and look of SQL. In order to make SPDBMS widely accepted, it is important to develop a high level query language as a wrapper for SP-Algebra. This high level language should look like the query language used for today's relational database management system and provide a comprehensive way to query SPDBMS. This led to the design and development of a new query language called semistructured probabilistic object query language (SPOQL), the high level structured query language for SPDBMS over the underlying SP-Algebra. SPOQL maps queries to SP-Algebra and acts as a wrapper to SP-Algebra. SPOQL is easy to learn and use when compared with SP-Algebra. SPOQL can be used to express simple to complex queries and can be mapped consistently with the underlying SP-Algebra. Also, SPOQL is designed and developed to express nested queries.

The following are my contributions through the current work for the development of SPOQL:

- Proposed and participated in the design of the grammar for SPOQL;
- proposed and participated in the design of the translation algorithm for mapping SPOQL queries to SP-Algebra queries;

- developed and implemented an SPOQL parser and translator for SPOQL queries which translates an SPOQL query into its equivalent SP-Algebra query using the proposed eager evaluation algorithm for SPOQL queries;
- designed and developed a new operation called “Mix Operation” to SP-Algebra, whose details are explained in the later parts of the report;
- enhanced the “selection” operation of SPDBMS to handle equijoin and implemented the enhancement whose details are explained in the later parts of the report.

The rest of the report is organized as follows.

Chapter 2 gives an overview of probabilistic databases, SPO model, SPDBMS, SP-Algebra, and the new SP-Algebra “mix operation” and enhanced SP-Algebra “select” operation. Chapter 3 describes the SPOQL syntax, semantics and translation algorithm. It explains the building of query trees and mapping of SPOQL queries to SP-Algebra and gives examples. Chapter 4 describes the architectural and component overview of SPOQL with all relevant implementation details. Chapter 5 describes the experimental evaluation of SPOQL in comparison to SP-Algebra queries. Chapter 6 describes conclusions and possible future work for extending the SPOQL language.

Chapter 2

Background and Related work

This section provides an overview of SPOs, a semistructured probabilistic object and SP-Algebra, algebra of atomic query operations on SPOs and the implementation of Semistructured probabilistic database management system (SPDBMS) in [20].

2.1 SPO model and SP-Algebra

SPO provides a flexible data structure to represent a probability distribution.

Definition 1 *A* ***Semistructured Probabilistic Object (SPO) S*** *is defined as a tuple* $S = \langle T, V, P, C, \omega \rangle$, *where*

$-$ *T is a relational tuple over some* semistructured schema $\mathtt{R}$ over $\mathtt{R}$. *T is referred to as the* **context** *of S*

$-$ $V = \{v_1, \ldots vq\} \subseteq \upsilon$ *is a set of* **random variables** *that participate in S and it is required that* $V \neq \emptyset$

$-$ $P : dom(V) \rightarrow \{(\text{u}$ *is the* **probability table** *of* ***S***. *Note that* ***P*** *need not be complete*

$-$ $C = \{(u_1, X_1), \ldots (u_s, X_s)\}$, *where* $\{u_1 \ldots, u_s\} = U \subseteq \upsilon$ *and* $X_i \subseteq dom(u_i)$, $1 \leq i \leq n$, *such that* $V \cap U = \emptyset$. *Here* ***C*** *is referred to as the* **conditional** *of* ***S***

$-\omega$, *called the* **path expression,** *is an expression of Semistructured Probabilistic Algebra (SP-Algebra)*

A collection $\{S_1 \ldots S_n\}$ of SPOs is called a SP-Relation. SPOs store probability distributions as follows. The **participating random variables** and **probability table** describes the actual distribution. The **conditional** part of SPO stores conditioning information for the distribution**. Context** part of SPO provides additional information supporting the probability distribution and can store known values of related parameters. Context variables are not considered as random variables. The **path** tells us how the SPO object is constructed. If the path is atomic (single unique identifier), then the object was *constructed from scratch* and inserted into the database. If the path is complex, then it indicates which database objects participated in its creation and what SP-Algebra operations have been applied to it.

Example: By using a SPO object, the conditional probability distribution of performance in a database class for CS majors who got ‘A’ in Data Structures can be represtented as follows.

ω*:* **S2**	
Major :”CS”	
Databases	Pr
A	0.3
B	0.3
C	0.2
D	0.1
E	0.1
Data Structures=’A’	

Figure 2.1 SPO Bird’s eye view

SP-Algebra, the query algebra for the SPO model, defines standard relational operations of selection, projection, Cartesian product and join. SP-Algebra also includes the conditionalization operation [20] more specific to the context of probabilistic databases. The current work extends the original SP-Algebra to include the ***mix operation*** and defines ***join conditions*** that can that can be applied to the SP-Algebra operations of Cartesian product, join and mix. The formal definitions of mix operation and join condition are provided in this section. The formal definition of remaining SP-Algebra operations are defined and described in detail in [20].

Informally, the SP-Algebra operations of SPOQL are explained as follows. In the definitions for **Atomic Projection** list, **Atomic Selection** conditions, and **Atomic Conditionalization** expression, let us consider that **`var, cnt, cnd,`** and **`tbl`** are the notations used to represent random variable, context, conditional and probability table parts of an SPO Object. Each of the above notational elements are referenced as [*Name.*]**`var,`**[*Name.*]**`cnt,`**[*Name.*]**`cnd,`**[*Name.*]**`tbl`** respectively. Here, the optional parameter *Name* represents the name of the SP-Relation. The definitions of Atomic Projection, Atomic Selection condition and Atomic Conditionalization are as follows:

Definition 2: Atomic projection list is defined as follows:

F ::= varlist | cntlist| cndlist

varlist ::= "var".$\langle name \rangle$ (, "var".$\langle name \rangle$)*,where $name \in \upsilon$

cntlist::= "cnt".$\langle name \rangle$ (,"cnt".$\langle name \rangle$)*,where $name \in \mathcal{R}$

cndlist::= "cnd". $\langle name \rangle$ (,"cnt".$\langle name \rangle$)*,where $name \in \upsilon$

Atomic selection conditions are described in Table 2.1

*Definition 3***: Atomic conditionalization** expression is defined as:

"var".$\langle name \rangle$ =Value, where $name \in \upsilon$

"var".$\langle name \rangle \in$ Value(,Value)*,where $name \in \upsilon$

Selection condition is inductively defined as:

Base: Atomic selection condition is a selection condition

Induction: Let c_1, c_2 be selection conditions. Then, the following are selection conditions: $c_1 \vee c_2$, $c_1 \wedge c_2$, $\neg c_1$

*Definition 4***: Join Condition** is defined as

$\langle Name \rangle$."cnt". $\langle name \rangle \langle Op \rangle \langle Name \rangle$."cnt".$\langle name \rangle$ where,

Name – name of sp-relation, $name \in \mathcal{R}$, and *Op*:= $, \leq, \geq, \neq, <, >$

Table 2.1 Atomic selection conditions

Type	**Expression**	**Explanation**
Context Condition	"cnt".$\langle name \rangle \langle Op \rangle$ constant "cnt".$\langle name \rangle \in T$	$name \in \mathcal{R}$ Op:$=, \leq, \geq, \otimes, <, >$
Variable Condition	"var".$\langle name \rangle \in V$	*name* – variable name where $name \in \upsilon$
Conditional Condition	"cnd".$\langle name \rangle$ =Value, "cnd". $\langle name \rangle \in C$ "cnd".$\langle name \rangle \in$ Value(,Value)*	*name* – variable name, where $name \in \upsilon$
Table Condition	"tbl".$\langle name \rangle$ = Value "tbl"."prob" Op RValue	Op:$=, \leq, \geq, \neq, <, >$ RValue $\in [0,1]$

Definition 5: **SP-Algebraic expressions** are inductively defined as:

Base: Let S be a name of an SP-Relation. S_a is an SP-Algebra expression

Induction: Let e_1 and e_2 be SP-Algebra expressions. Let *c* be a Selection Condition (SC), *f* be a Projection List (PL), *d* be a conditionalization expression (CE) and g be a join condition (JC). The SP-Algebra expressions are described in Table 2.2.

Selection(σ_c(S)) operation finds SPOs in a SP-Relation that satisfy a specific selection condition. The selection operations on *context, participating variables* or *conditionals* do not alter the content of the selected objects (SPOs) and result in either an SPO being selected or not in its entirety, if in classic relational algebras. If the selection operations are used either on the values of the random variable or probabilities or probability table, then the resultant SPO will contain only those probability table rows that match selection condition but retains the context, participating variables and conditional information. The query can be expressed in English as follows: "Find all those probability distributions in which variable VOP has value success with probability greater than 0.5”

Table 2.2 SP-Algebra expressions

Type	**SP-Algebra expression**
Selection	$\sigma_c(e_1)$
Projection	$\pi_f(e_1)$
Conditionalization	$\mu_d(e_1)$
Cartesian product(CP)	$e_1 \times e_2$
CP with join condition	$e_1 \times_g e_2$
Join	$e_1 = \times\ e_2$, $e_1 \times =\ e_2$
join with join condition	$e_1 = \times_g\ e_2$, $e_1 \times =_g\ e_2$
Mix	$e_1 \otimes e_2$
mix with join condition	$e_1 \otimes_g e_2$

Projection (π_f (S)) is an operation that simplifies SPOs. The projection operations used on *context* and *conditionals* variables are similar to the classic relational algebra, meaning either a context or conditional is removed from the resultant SPO objects depending on which of these two variables the projection is applied. The rest of the resultant SPO does not change otherwise. But, the projection operation on the set of *participating random variables* is a delicate operation as it corresponds to removing the other random variables from consideration in a joint probability distribution. The result of this operation is a new *marginal probability distribution* and is stored as the probability table component of the resultant SPO. This marginal probability distribution is obtained in two steps. First, the columns for random variable that are to be projected are removed from the probability table. Now, the probability table may contain duplicate rows whose values for all the fields except probability values coincide. In the second step, all the duplicate rows of same type are collapsed(coalesced) into one, with a new probability value computed as the sum of values in the collapsed rows.

Conditionalization (μ_d (S)) is the operation that is used for conditioning the joint probability distribution. This operation is applied on conditional variables and is performed in two steps. First, it removes all the rows from the probability table of the SPO that do not match the condition. Then the conditional variable column give in condition is removed from the table and the remaining rows are coalesced if needed in the same way as in projection operation and then the probability values are normalized.

Cartesian product ($\times$) and **join** ($_=\times$, $\times_=$) are the operations used to build a joint probability distribution from the input SPOs. The join operation is applicable to those SPOs having common participating random variables and Cartesian product is applicable to those SPOs with disjoint lists of participating variables. Two SPOs are *cp-compatible,* if their participating variables are disjoint and their conditionals coincide. Two SPOs are join-compatible if their participating variables are not disjoint and their conditionals coincide.

Mix($\otimes$) operation is a new operation introduced to SP-Algebra along with the development of SPOQL. This operation also constructs a joint probability distribution from the input SPOs. Consider two SP-Relations S and S^1. They can be either join compatible, cp-compatible, or neither join nor cp-compatible. The mix operation is the union of the join and Cartesian product. Let $S = \langle T, V, P, C, \omega \rangle$, $S^1 = \langle T^1, V^1, P^1, C^1, \omega^1 \rangle$ are two cp-compatible or join-compatible SPOs. Then mix operation $S \otimes S^1$ is defined as follows:

$S \otimes S^1 = (S \times S^1) \cup (S _=\times S^1)$

The current work also contributes to the implementation of the mix operation and this operation is effectively used by the SPOQL translation algorithm as and when needed. The current works also extends combination operations such as Cartesian product, mix and join with join conditions. That means, SPOs can now be joined based on the relationships of their respective context attributes. Besides satisfying the conditions that are applicable to SPOs participating in Cartesian product mix and join operations, their context elements should satisfy the join condition specified. Otherwise, elements that do not satisfy the join condition are not combined.

2.2 SP-Algebra Equivalences

Zhao [20] established the SP-Algebra equivalences in preparation for query optimization for SP-Algebra. They are shown in Table 2.3 and Table 2.4. We have established additional equivalences involving join and Cartesian product operators. In table 2.3, SC,PL and CE represents selection condition, atomic projection list and atomic conditionalization expression respectively. Selection condition, atomic projection list and atomic conditionalization expression definitions can be seen in earlier section.

Table 2.3 Query Equivalences for SP-Algebra operations

Equivalence	**Condition**
$\sigma_{c \wedge c1}(e_1) \equiv \sigma_c(\sigma_{c1}(e_1))$	c and c^1 are SCs
$\sigma_c(\sigma_{c1}(e_1)) \equiv \sigma_{c1}(\sigma_c(e_1))$	c and c^1 are SCs
$\pi_{f \cap f1}(e_1) \equiv \pi_f(\pi_{f1}(e_1))$	f and f^1 are PLs
$\pi_f(\pi_{f1}(e_1)) \equiv \pi_{f1}(\pi_f(e_1))$	f and f^1 are PLs
$\mu_{d \wedge d1}(e_1)) \equiv \mu_d(\mu_{d1}(e_1))$	d and d^1 are CEs
$\mu_d(\mu_{d1}(e_1)) \equiv \mu_{d1}(\mu_d(e_1))$	d and d^1 are CEs
$e_1 \times e_2 \equiv e_2 \times e_1$	e_1 and e_2 are cp-compatible
$(e_1 \times e_2) \times e_3 \equiv e_1 \times (e_2 \times e_3)$	e_1, e_2 and e_3 are cp-compatible

Table 2.4 Query Equivalences for SP-Algebra operations

Equivalence	Condition
$e_1 =\times (e_2 \times e_3) \equiv (e_1 =\times e_2) \times e_3$	e_1 , e_2 are join compatible and e_2 , e_3 are cp-compatible
$e_1 \times= (e_2 \times e_3) \equiv (e_1 \times= e_2) \times e_3$	e_1 , e_2 are join compatible and e_2 , e_3 are cp-compatible
$e_1 \times (e_2 =\times e_3) \equiv (e_1 \times e_2) =\times e_3$	e_1 , e_2 are cp-compatible and e_2 , e_3 are join-compatible
$e_1 \times (e_2 \times= e_3) \equiv (e_1 \times e_2) \times= e_3$	e_1 , e_2 are cp-compatible and e_2 , e_3 are join-compatible
$e_1 =\times e_2 \equiv e_2 \times= e_1$	e_1 and e_2 are join-compatible P(X,Y)*P(Z|Y) = P(X|Y)*P(Y,Z), where X,Y are variables from e_1 and Y,Z are variables from e_2

Selection on probabilities in general does not commute with other operations. This is because the projection, conditionalization and Cartesian product /join/mix operations change the probability distribution table stored in the probability table of the resultant SPO.

2.3 Implementation of SPDBMS

SPDBMS is implemented on top of a relation database management system using Java. The Figure 2.1 depicts the overall architecture of the original system [20]. The SPDBMS application server processes query request like standard database management instruction and SPOQL queries from a variety of applications.

The application server provides a JDBC-like API, through which client applications can send standard database management instructions, such as CREATE

DATABASE, DROP DATABASE, CREATE SP-RELATION, DROP SP-RELATION, INSERT INTO SP-RELATION, DELETE FROM SP-RELATION, as well as SP-Algebra queries to the server. Our SPOQL implementation has been integrated into the architecture shown in Figure 2.1.

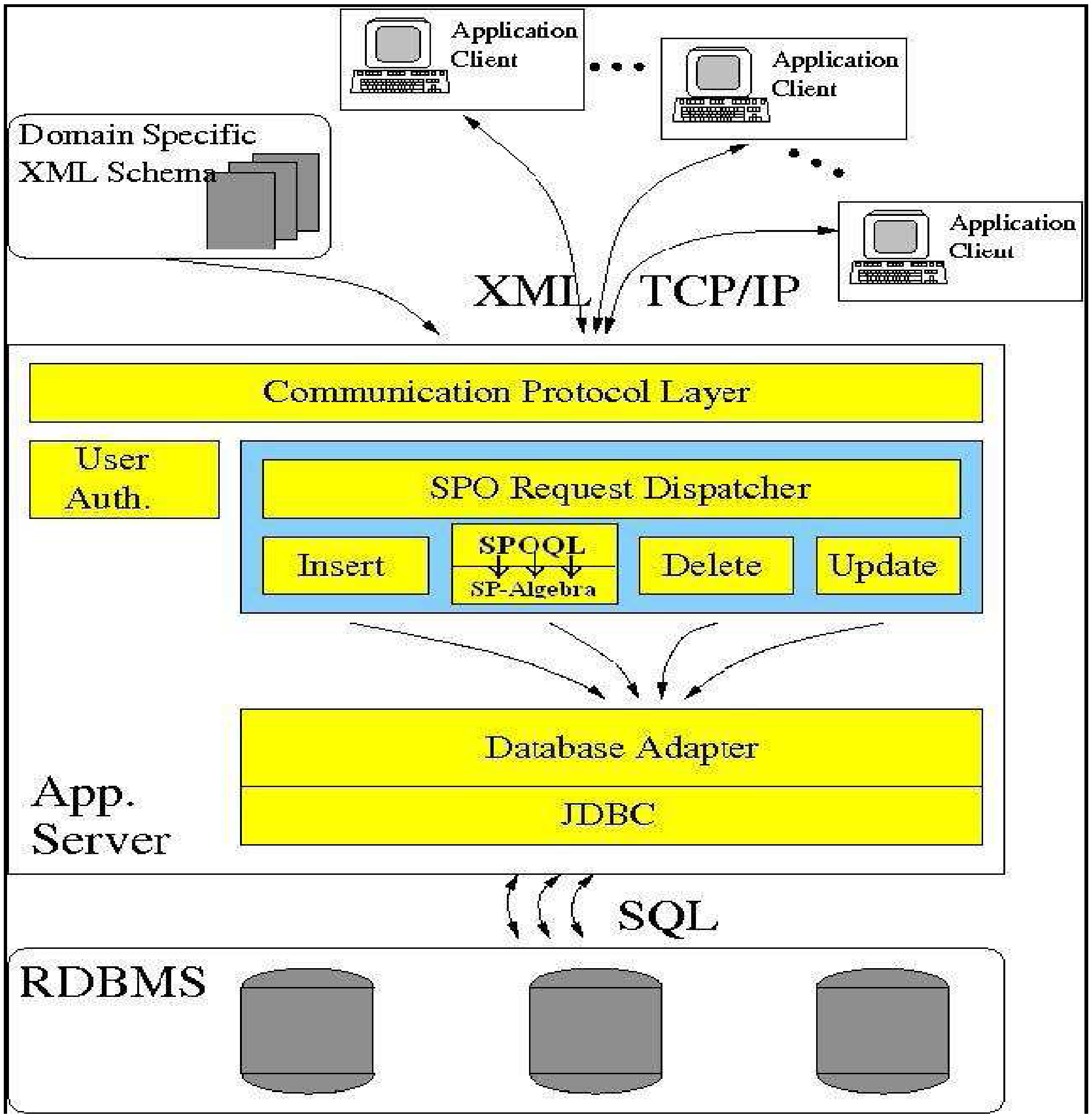


Figure 2.2 The overall architecture of SPDBMS

2.4 Related Work

Significant research has been carried out in the management of probabilistic data over the years. The early relation models proposed to store and process probabilistic data [12,2,6,1] have been replaced by object oriented [11,8] and semistructured models [10,14,4,20]. The work of Cavallo and Pittarelli [2] extended the relation model to represent uncertainty in information using probabilistic calculus. Every tuple in the relation is assigned a probability measure and it indicates the join probability of all the attribute values in the tuple. Barbara et al. [1] proposed an extension of the relational model using probability theory by adopting a non-1NF probabilistic data model. First normal form (1NF) is a normal form used in database normalization. First normal form excludes the possibility of repeating groups by requiring that each field in a database hold an atomic value, and that records be defined in such a way as to be uniquely identifiable by means of a primary key. They redefined the projection, selection and join operations using semantics of probability theory and have also introduced a new set of operators to explain various set of possibilities. Dey and Sarkar [6] proposed a probabilistic database framework with relations adhering to first normal form (1NF). In this model, the probability measure assigned to every tuple indicates the joint probability distribution of all the no-key attributes in the relations.

They proposed a closed form query algebra and introduced conditionalization operation in the context of probabilistic model. Also, the proposed a non-procedural probabilistic query language called PSQL [7] as an extension of the SQL language. The ProTDB [14] proposed by Nierman, et al. is close to the SPDBMS approach. In ProTDB, XML data model is extended by associating a probability to each element with the modification of regular non-probablisitic DTDs, and independent probabilities are attached to each individual child of an object. The probabilities in an ancestor-descendant chain are related probabilistically, resulting in conditional probabilities in XML documents. Some drawbacks of ProTDB are overcome by the PXml framework proposed by Hung, Getoor and Subrahmanian [10]. PXml supports arbitrary distributions over sets of children and allows arbitrary acyclic dependency models. They also provide that for any query in their model there is a mapping to an equivalent query in the Bayesian network. Bayesian networks are directed acylic graphs whose nodes represent variables, and whose arcs encode the conditional dependencies between the variables. They also proposed a probabilistic interval XML data model, PIXml [9]. But joint probability distributions cannot be represented conveniently by either PXml or PIXml models. The work on SPDBMS [20] combines and extends the ideas in these papers and applies them to an SPO model. The data stored in the SPDBMS does not conform to a rigid schema.

Chapter 3

Semistructured Probabilistic Object Query Language

3.1 Syntax of SPOQL

As stated earlier, SPOQL is designed as a high level query language for SPDBMS with declarative syntax. This section explains the syntax and semantics of SPOQL as well as the translation algorithm that maps the query expressed through SPOQL into its corresponding SP-Algebra. The main objective of this work is to design a query language that can handle all the queries expressed through sp-algebra with SPOQL and to translate them with a consistent mapping mechanism into their corresponding SP-Algebraic expression.

SPOQL is designed to handle simple to complex queries to as well nested queries. The basic syntax of SPOQL query looks like as follows:

SELECT <selectlist>
 FROM <fromlist>
 [**WHERE** <condition>]
 [**CONDITIONAL** <conditionlist>]

SPOQL query relates to its sp algebraic expression and consists of selections, projections, conditionalizations and combining operations such as mix, Cartesian products and joins. Each SPOQL operates on one or more SP Relations and returns an SP-Relation. An SP-Relation is a collection of one or more SPOs

3.2 SPOQL Query Parts

Each SPOQL query is made up of two to four clauses. All SPOQL queries must have SELECT and FROM clauses whereas WHERE and CONDITIONAL clauses are optional. The SELECT clause consists of a list of SPO variables such as context, conditional and random variables to be projected and retained in the resultant SP Relation. The FROM clause consists of a list of participating SP Relations along with their combining operations. The optional WHERE clause consists of a list of selection conditions on the SP-Relations specified in the FROM clause as well as the join conditions on the combining operations specified in the FROM Clause. The optional CONDITIONAL clause specifies the conditionalization expressions on the SP-Relations specified in the FROM Clause. Each of these clauses is explained in details as follows.

selectlist: This is a sequence of random variables, context variables and conditional variables that are participating in the projection operation and every variable corresponds to an SP-Relation in the *fromlist.* SPOQL also allows wildcard operation '*' in the absence of any selectlist and in this case the entire resultant object is retained in the result. Wild card operation can be applied to each of context, conditional and random variables in selectlist.

fromlist: This is a sequence of SP-Relations separated by combining operations "TIMES", "JOIN" and ",". Here, "TIMES" stand for Cartesian product, "JOIN" for the join operation and "," for the mix operation. SPDBMS by default allows left join among the participating SP-Relations. SPOQL also provides scope for nested queries. SPOQL facilitates this by allowing SPOQL queries in its fromlist. SPOQL also facilitates for specifying associativity of combination operation by enclosing the participating SP-Relations and provides feature for aliasing this operation.

Condition This is a sequence of selection and join conditions separated by the keyword "AND". Each selection condition corresponds to an SP-Relation in the fromlist.

Similarly, each join condition corresponds to a combing operation from the fromlist depending on the SP-Relations provided by it.

3.3 SPOQL Semantics

ω*:* **S1**	
Work- type:nursing City:Lexington	
VOP	P
success failure	0.75 0.25

A=high G=high
S=high WR=low
C=high	

(a)	

ω*:* **S2**	
City:Lexington	
WR	P
high low	0.8 0.2
WH=good	
(b)	

Figure 3.1 Probability Distributions

Consider the following few simple SPOQL queries.

1. **SELECT * FROM** S1
   WHERE S1.tbl.VOP='success'
   AND S1.tbl.prob >0.7

2. **SELECT** S1.cnt.city, S1.cnd.A **FROM** S1

3. **SELECT * FROM** S2 **JOIN** S3

4 .**SELECT * FROM** S4
CONDITIONAL S4.var.WR=’high’

ω: **S3**	
S	P
high low	0.7 0.3
WH=good (c)	

ω: $\sigma_{tbl.prob>=0.75}$**(S1)**	
work-type: nursing city:Lexington	
VOP	P
success	0.75

A=high G=high S=high
WR=low C=high
(d)	

ω: π_f**(S1)**	
city:Lexington	
VOP	P
success failure	0.75 0.25
A=high (e)	

Figure 3.1 (continued)

SPO(s) in Figure 3.1(d), Figure 3.1(e), Figure3.1 (f), Figure3.1 (g) can be obtained using the above SPOQL queries. These are simple queries that represent a single SP-Algebra operation. The following is an example of a SPOQL query representing multiple SP-Algebra operations.

ω**: S4=S2 X S3**		
city: Lexington		
WR	S	P
high	high	0.7
low	high	0.3
high	low	
low	low	
WH=good		
(f)		

ω: $\mu_{WR=high}$**(S4)**	
city: Lexington	
S	P
high	0.7
low	0.3

WH=good
WR=high	

(g)	

Figure 3.1 (continued)

SELECT * FROM S2 JOIN S3
 WHERE S2.tbl.WR='high'
  AND S3.tbl.prob<0.7

This query involves one join operation and multiple selection operations. This query raises questions about its corresponding SP-Algebra translation and the order in which these different operations are evaluated. One interesting case to consider is that, in classical relational algebra, Cartesian product and join operations commute and this feature allows determining the execution plan during the optimization stage irrespective of the order of operations produced by SQL Parsers. But SP-Algebra deals with probabilistic data. Cartesian product and join operations and selection on probabilities and probability tables do not commute. Hence, it is essential to determine the order of SP-Algebra operations and generate an execution plan after parsing the SPOQL query and before translating the SPOQL query into its corresponding SP-Algebraic expression. In order to have consistent query evaluation, with consistent reasoning, the following order of precedence is established for every SP-Relation belonging to the *fromlist.*

1. Conditionalization operation using *conditionlist*
2. Selection operation using selection conditions from *condition*
3. Projection operation using *selectlist*
4. Join/Times/Mix operation using combining operations from *fromlist* and join conditions from *condition.*

In addition to precedence rules, there are other SP-Algebra translation issues that need to be handled. SPOQL provides nesting and aliasing in a query to provide flexibility or to override the precedence order. In order to ensure proper query translation, a separate scope for each level of nesting with in SPOQL query is defined. This scoping rule does not permit any elements from the *selectlist, condition* and *conditionlist* to address SP-relations from the fromlist not belonging to the same query level. For example consider the following SPOQL query Q.

SELECT * FROM
(**SELECT * FROM** S2 **WHERE** S2.cnt.year=2000), S1
WHERE S1.cnt.age='19-20' **CONDITIONAL** S2.var.LY='A'

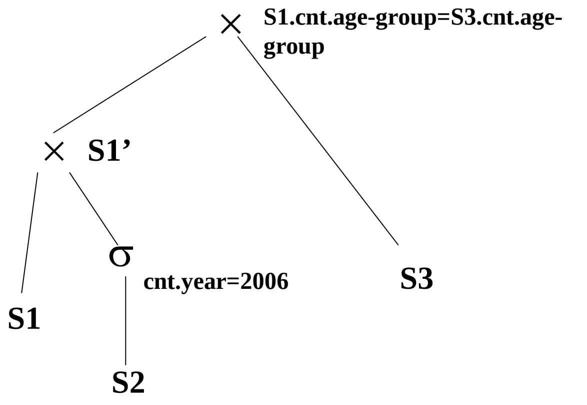
According to the above scoping rule, the above SPOQL query Q is semantically invalid because the conditional operation on SP-relation S2 is present at a different level. The above SPOQL query Q can be expressed correctly as follows.

SELECT * **FROM**
(**SELECT** * **FROM** S2 **WHERE** S2.cnt.year=2000 **CONDITIONAL** S2.var.LY='A')
, S1 **WHERE** S1.cnt.age='19-20'

The next SP-Algebra translation issue to consider is the order in which SP-relations from the *fromlist* are combined. This does not apply in the absence of any join conditions in the query and the classical left-to-right evaluation can be performed. However, the presence of join conditions in a SPOQL query requires special handling. For example, consider the following SPOQL query Q'.

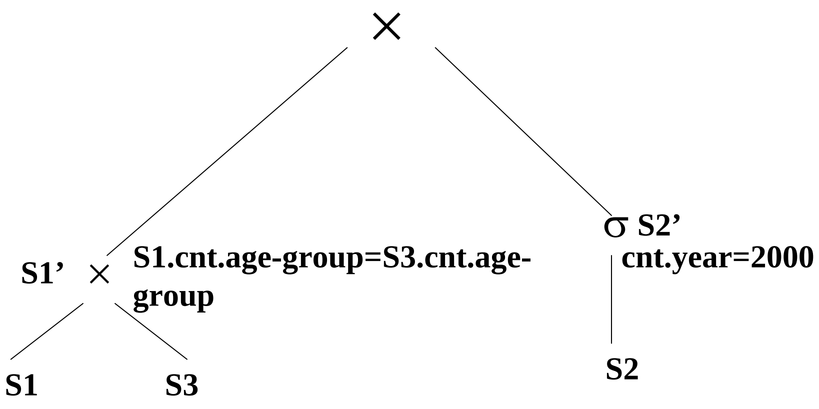
SELECT * FROM S1 **TIMES** S2 TIMES S3
WHERE S2.cnt.year=2006 **AND** S1.cnt.age-group=S3.cnt.age-group

If the traditional left-to-right evaluation of the *fromlist* is performed, then the resultant query evaluation is show in Figure 3.2 (a) The actual query evaluation we intend to express through query Q' is represented in Figure 3.2 (b)



Query tree for SPOQL query Q' based on left-to-right evaluation only

Figure 3.2(a) Query tree for SPOQL query Q' based on left-to right evaluation



Query tree for SPOQL query Q' based on left-to-right evaluation with join condition(s) priority

Figure 3.2 (b) Query tree for SPOQL query Q' based on left-to-right evaluation with join conditions(s) priority

In order to override this ambiguity, we must ensure that SPOQL query parser/translator will override the default order of combining SP-relations in the *fromlist* of query Q' and other similar SPOQL queries. This feature can be verbalized as *materialize the combination of SP-relations under join conditions first.* SPOQL also introduces a new feature for explicitly specifying the order in which SP-relations in the *fromlist* can be combined. The explicit ordering query Q' can be now expressed with this feature as follows.

SELECT * FROM S2 **TIMES** (S1 **TIMES** S3)
WHERE S2.cnt.year=2006
AND S1.cnt.age-group=S3.cnt.age-group

The query tree for this explicit order query Q' is same as in Figure 3.2 (b). Identical pairs of SP-relations in both orderings can be identified by applying any join conditions that refer to the same explicit ordering pair of SP-relations. Hence, the SPOQL query evaluation algorithm builds by applying precedence rules and then determining the order of join, Cartesian or mix operations with SP-relations as the leaves.

3.4 SPOQL Query Translation Algorithm

The translation algorithm of SPOQL consists of following two steps.

1) Building the query tree
2) Translation of the query tree into SP-Algebra

The algorithm takes SPOQL query Q as its input and produces a query tree as its output representing the SP-Algebraic expression describing the semantics of query Q. The algorithm is described as follows:

```
Evaluate(SPOQL query Q)
    1. Validate *Q* to check for scoping consistency and non duplicate SP-relations in the
       *fromlist*
    2. If valid, then next step, else “Semantic error”.
    3. Extract *fromlist* , *selectionlist*, *condlist*, *projlist* from Q
    4. Let *fromlist*=($E_1$,...,$E_k$)
    5. for i=1 to  do
           $T_i$ =Build-subtree($E_i$, *selectionlist, condlist , projlist)*
    6. T= $T_1$
       for i=2 to k do
       *T*=Combine(T, $T_i$)
    7. return Build-path(T, selectionlist ,condlist, projlist)
```

```
Build-subtree(fromExp, selectionlist, condlist, projlist)
    1. if *fromExp* is a SPOQL query
    return Evaluate(fromExp)
    2. else
      if *fromExp* is an SP-Relation
      return Build-path(fromExp, selectionlist, condlist, projlist)
     else
       Let fromExp =($E_1$ Op $E_2$)Name
         $T_1$= Build-subtree($E_1$, selectionlist, condlist, projlist)
          $T_2$=Build-subtree($E_2$, selectionlist, condlist, projlist)
           T=Op($T_1$, $T_2$)
          if Name=”””” return T
              else return Build-path(T, selectionlist, condlist, projlist)
```

Figure 3.3 Algorithm for translating SPOQL queries into SP-Algebra expressions

```
Build-path(fromExp, selectionlist, condlist, projlist)
    1. Perform conditionalization operation on fromexp using condlist
    2. Perform selection operation on resultant fromexp using selectionlist
    3. Perform projection operation on resultant fromexp using projlist
    4. return resultant SP-Relation tree T'
```

Figure 3.3 (Continued)

In this algorithm, **Build-path ()** is a routine that computes the conditionalization-selection-projection subtree for each SP-Relation / SP-Relation alias. Build-subtree **()** is a function that produces the query sub tree for a single *fromlist* entry (either an SP-Relation, or a nested SPOQL query or a nested join/product/mix operation).

The working of the algorithm is explained as follows with sample SPOQL queries.

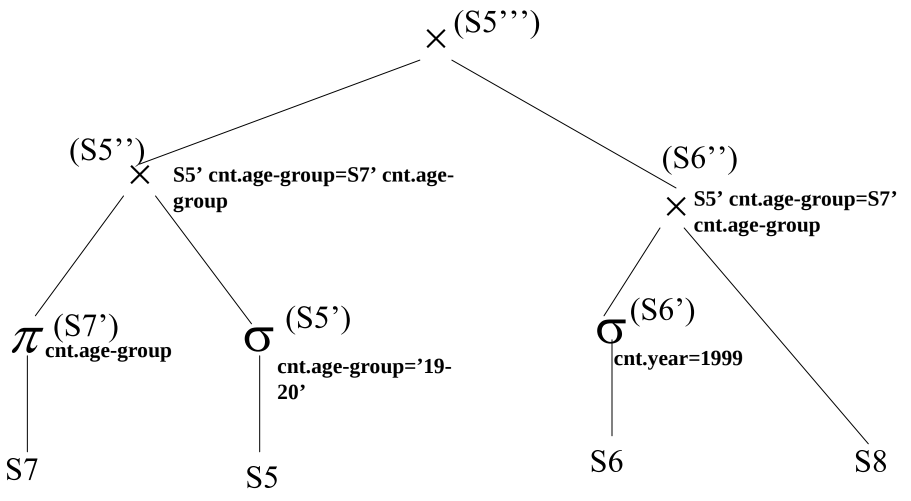
Example 3.1:
SELECT * FROM S
WHERE cnt.city="Lexington" **AND** var.Y **IN V**
AND tbl.prob>0.2

The above query is a conjunction of atomic selection conditions. This produces the following query tree representing the SP-Algebraic expression. The order of selection conditions in the SPOQL query does not matter due to the SP-Algebra properties given in Table 2.4

Example 3.2:
Consider the following SPOQL query Q'' where S5, S6, S7, S8 represent the SP-Relations that describe the distribution for the Welfare to Work client's characteristics-Aptitude, Goals, Confidence, and work history respectively. Welfare to Work [3] is a research project aimed at developing software tools to support Welfare to

Work case managers in advising their clients. The Welfare to Work case managers have to deal with decision-making with uncertainty and constraints in client's charactersistics. Hence, the client characteristics can be easily represented as an SPO and can be queried using SPOQL.

SELECT S7.cnt.age-group FROM S5 TIMES S6 TIMES S7 TIMES S8 WHERE S6.cnt.year = 1999 AND S5.cnt.age-group=S7.cnt.age-group AND S5.cnt.age-group='19-20' AND S6.cnt.year = S8.cnt.year



× (S5''')
(S5'')
× S5' cnt.age-group=S7' cnt.age-group
(S6'')
× S5' cnt.age-group=S7' cnt.age-group
π (S7') cnt.age-group
σ (S5') cnt.age-group='19-20'
σ (S6') cnt.year=1999
S7
S5
S6
S8

Figure 3.4 Query tree for Example 3.2

Initially, the SPOQL query is evaluated for its syntactic and semantic validity. Then, according to the build path algorithm the selection operation on context is performed on SP-Relations S5 and S6 resulting in SP-relations S5' and S6' respectively. The projection operation on context is then performed on SP-Relation S7 resulting in SP-Relation S7'. Then according to Build-subtree algorithm, SP-relations S5' and S7' are combined by applying the respective join condition to the Cartesian product resulting in SP-relation S5''. Similarly SP-relations S6' and S8 are combined in resulting SP-relation S6''. The resulting SP-relation S''' is the Cartesian product of SP-Relations S5'' and S6''.

Example 3.3:

SELECT * FROM S1 **JOIN** (S2, S3) **WHERE** S1.var.A in V **AND** S2.tbl.prob >0.2

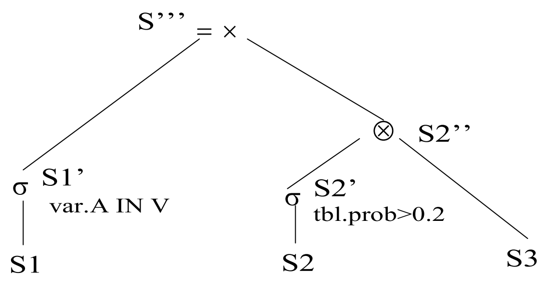


Figure 3.5 Query tree for Example 3.3

This query illustrates the evaluation of explicitly ordered SP-relations in a SPOQL query. The above query is evaluated for its syntactic and semantic validity. According to the Build-path algorithm, the selection operation on Variable is performed on SP-Relation S1 and selection operation on probability value is performed on SP-Relation S2 resulting in SP-relations S1' and S2'. Then according to Build-subtree algorithm, SP-relations S2' and S3 are combined using mix operation resulting in SP-relation S2''.Thus, the resulting SP-Relation S''' is the join operation of SP-relations S1' and S2''.

Example 3.4:

SELECT cnt.age **FROM** S where tbl.prob>0.5 **CONDITIONAL** var.LY=B

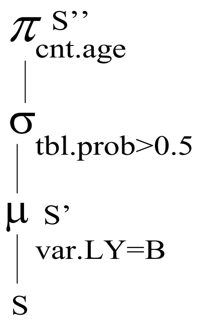


Figure 3.6 Query tree for Example 3.4

This is a simple query that illustrates the order of evaluation in a SPOQL query. According to build path algorithm, Conditionalization operation is performed on the SP-Relation S resulting in SP-relation S'. Then, selection operation on probability table value is performed on SP-Relation S' resulting in SP-relation S''. And finally, projection operation on context is performed on SP-relation S'' resulting in the final output SP-relation S'''.

Example 3.5:

SELECT * FROM S1 **JOIN** (**SELECT** * from S2 **TIMES** S3 where S2.cnt.year=2006 **CONDITIONAL** S3.var.LY=A) **WHERE** S1.cnt.DA=18

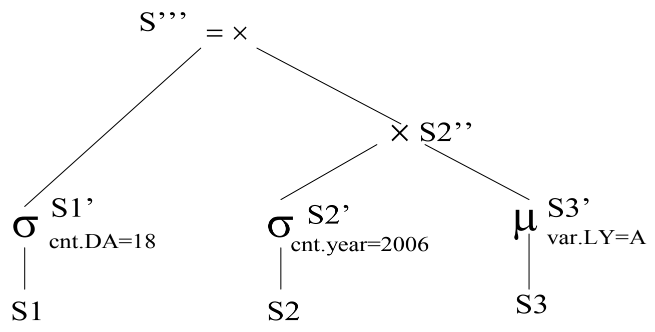


Figure 3.7 Query tree for Example 3.5

This query illustrates the functionality of nested queries supported by SPOQL. The above query is evaluated for its syntactic and semantic validity. According to build path algorithm, conditional operation on conditional variable is performed on SP-relation S3 resulting in SP-relation S3’ and selection operation on context is performed on SP-relation S2 resulting in SP-relation S2’. Then according to Build-subtree algorithm, SP-relations S2’ and S3’ are combined by applying the Cartesian product resulting in SP-relation S2’’. Now, according to Build-path algorithm, selection operation is performed on SP-relation S1 resulting in SP-relation S1’. And then according to Build-subtree algorithm, SP-relation S1’ and S2’’ are combined by applying the join operation resulting in the final output SP-relation S’’’.

Chapter 4

Architecture of Semistructured probabilistic Query Language

4.1 System Architecture

Given as a basis the translation mechanism and query language described in the previous section, we now introduce the SPOQL translator system architecture and discuss the interaction between components of the system.

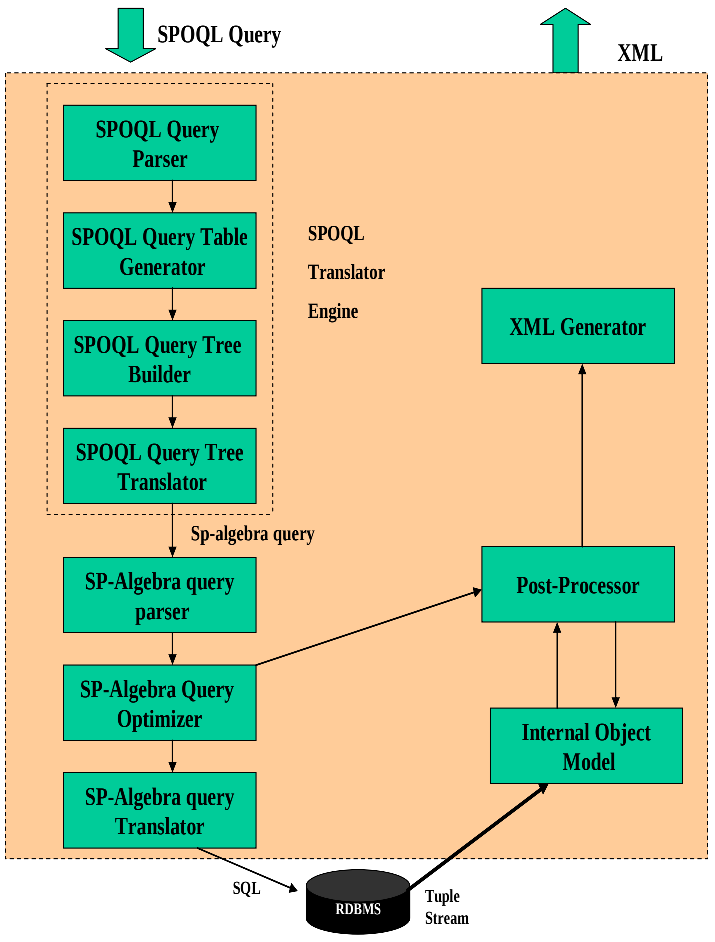


Figure 4.1 Architecture of SPOQL integrated into SPDBMS

The basic architecture of SPOQL translator system and its integration into SPDBMS is shown in Figure 4.1. SPDMBS is the backend storage of probabilistic data for various applications that are developed for the Welfare-to-Work modeling project [3]. These applications are currently interacting with SPDBMS using SP-Algebra queries. Now with the implementation of SPOQL, these applications can start querying SPDBMS using SPOQL queries. The ***query compilation*** layer of the SPOQL system consists of the ***parser***, ***query table generator***, ***query tree generator*** and ***query tree translator***.

The SPOQL parser accepts a textual representation of SPOQL query from various applications, transforms it into a parse tree, and then passes it to the query table generator for building a transposition table. This transposition table built is then passed to query tree generator. The query tree generator evaluates the SPOQL query represented in a transposition table. Then it builds a logical query evaluation plan and manipulates it as per the semantics of SPOQL. Then the logical query evaluation tree is passed to the query tree translator to translate the SPOQL query into its corresponding SP-Algebraic expression. The SP-Algebra expression is then finally executed by the rest of the SPDBMS components.

4.2 Component Overview

4.2.1 Query Parser

In this step, the query is parsed as per the grammar rules of SPOQL. If the query is parsed successfully, then the query is validated against the validation rules of SPOQL. If the query does not pass these two steps successfully, the SPOQL engine throws the corresponding error to the calling application and stops further processing of SPOQL query. Otherwise, the constructed parse tree is passed to the next step for further evaluation. The Java parser API is used to build the SPOQL parser[22].

4.2.2 Query Table Generator

In this step, the SPOQL query is analyzed and a proper transposition table is constructed for further evaluation in subsequent stages. As stated in an earlier chapter, the

SPOQL engine is designed to handle nested queries. To handle this, the SPOQL engine employs the strategy resembling depth-first search for analyzing the query. And since depth first search is used for analyzing SPOQL queries, the engine needs to keep track of nested queries analyzed so far. Otherwise, the engine will end up analyzing the same nested query again and again. In order to overcome this problem and to represent the SPOQL query through a proper data structure for further handling, the SPOQL engine uses a transposition table. This table is a hash table of each of the nested queries (inner level queries) as well as top level query completely analyzed from left to right. A hash table is a kind of data structure that associates keys with values. In this case, the SPQOL engine pairs nested query/top level query with its corresponding list of query parts such as *fromlist, selectlist , condition, conditional.* SPOQL engine generates the keys relative to the position of the nested query in the SPOQL query. Starting with the top level query, the keys are numbered as $PS, $NS1, $NS2, ….. , $NSn where $PS is the key associate with top level select statement and $NS1, $NS2,…, $NSn are keys associated with corresponding nested select statements if are present.

4.2.3 Query Tree Generator

By using a transposition hash table constructed for the SPOQL query, the query tree generator produces the query evaluation tree for the query as per the semantics of SPOQL. The query tree is represented using a stack data structure. A typical entry in the stack consists of a SPOQL operator to be applied, SPOQL relation(s) and join or select conditions if are present. The query tree generator uses a recursion technique for evaluating select statements at various levels. The query tree is constructed by evaluating the query from left to right. The inner most SELECT statement is evaluated before evaluating the SELECT at its preceding upper level. The precedence of operations and scoping rules defined in the previous chapter are applied while constructing the query tree. The query tree is used by the query tree translator for translating into SP-Algebraic expression.

4.2.4 Query Tree Translator

This is the final step in the translation of SPOQL query into SP-Algebra expression. The query tree is represented as a stack. The translator takes out each item from the stack and translates each atomic SPOQL query expression into its corresponding SP-Algebraic expression. All these atomic SP-Algebraic expression are appended to form the SP-Algebraic representation of the given SPOQL query.

Chapter 5

Experimental Results and Analysis

This section explains the experimental setup used for testing SPOQL. The results of the tests conducted by writing SPOQL queries for SPDBMS are then compared with those results obtained by writing straight SP-Algebra queries.

System Environment

The current system has the application server and the database server running on the different machines. Hence a network delay is possible during these tests. The test datasets used in the experiment are those used for testing SP-Algebra queries. In order to accommodate network delay during these tests for comparative analysis, the queries were initially run by the SP-Algebra server and then later by SPOQL Server. Oracle 8i was selected as the backend database server for the current system. The current system has 440 M Hz Sun Ultra 10 running Solaris OS with 1GB of main memory. For all the experiments conducted, 256 M memory is allocated for the JVM and the timing is done on the server side. The execution time for SP-Algebra query and SPOQL query is captured. Then, the translation time from SPOQL to SP-Algebra is calculated as the difference of these times. The results are shown in Table 4.2. The tests are run for SPOs with 2, 3 and 4 variables.

From the results obtained, we can see certain cases where the execution time of SPOQL query is smaller than that of its equivalent SP-Algebra query. This is because we ran the tests for the set of queries written in SP-Algebra using SP-Algebra server initially and then ran the same set of queries written in SPOQL using SPOQL Server against the same set of SP-Relations in SPDBMS. As SPDBMS is implemented on top of RDBMS, RDBMS has the ability to cache query evaluation plans for queries for later use. Hence, if the same query is submitted once again then RDBMS uses the cached execution plan and fetches results faster.

Table 5.1 Execution time of SPOQL queries and SP-Algebra queries for SPOs of variable size 2

Number of SPOs	Query Type	Execution time for SP-Algebra (msec)	Execution time for SPOQL (msec)	Difference(SPOQL time - SP-Algebra time) (msec)
100	Select on context	78	60	-18
100	Select on conditional	39	58	19
100	Select on variable	57	38	-19
100	Select on table	82	79	-3
100	Project on context	100	101	1
100	Project on conditional	80	56	-24
100	Project on variable	69	58	-11
100	Conditionalization	53	53	0
100	Select on Probability value	205	243	
100	Cartesian product	4638	4837	38
100	Join	297	326	29
100	Mix operation	4630	4646	16
100	Complex query 1	54	115	61
100	Complex query 2	55	103	48
100	Complex query 3	136	220	84

Table 5.2 Execution time of SPOQL queries and SP-Algebra queries for SPOs of variable size 3

Number of SPOs	Query Type	Execution time for SP-Algebra (msec)	Execution time for SPOQL (msec)	Difference(SPOQL time - SP-Algebra time) (msec)
100	Select on context	17	39	22
100	Select on conditional	11	20	9
100	Select on variable	47	59	12
100	Select on table	48	53	5
100	Project on context	98	87	-11
100	Project on conditional	46	42	-4
100	Project on variable	44	43	-1
100	Conditionalization	43	43	0
100	Select on Probability Value	116	117	1
100	Cartesian Product	18236	18283	47
100	Join	983	986	3
100	Mix operation	18811	19076	265
100	Complex query 1	78	47	-31
100	Complex query 2	65	61	-4
100	Complex query 3	185	201	16

Table 5.3 Execution time of SPOQL queries and SP-Algebra queries for SPOs of variable size 4

Number of SPOs	Query Type	Execution time for SP-Algebra (msec)	Execution time for SPOQL (msec)	Difference(SPOQL time - SP-Algebra time) (msec)
100	Select context	18	40	22
100	Select on conditional	13	20	7
100	Select on variable	53	68	15
100	Select on table	49	68	19
100	Project on context	168	145	-23
100	Project on conditional	43	45	2
100	Project on variable	19	46	27
100	Conditionalization	23	51	28

Table 5.3 (continued)

100	Select on Probability Value	39	64	25
100	Complex query 1	35	53	18
100	Complex query 2	95	141	46
100	Complex query 3	260	397	137

From the tables we can observe that the translation time is relatively greater for complex queries and not a major overhead for simple queries as well for join, Cartesian product and mix operations.

Chapter 6

Conclusions and Future work

The SPO model for management of uncertain data in databases provides flexibility for storing and manipulating large collections of probability distributions. SP-Algebra, the original query language for SPDBMS provides all major database operations and introduces some operations, such as conditionalization specific to probabilistic database management system. The traditional limitations of SP-Algebra — the functional syntax and unfamiliarity to application programmers have been alleviated by SPOQL. SPOQL is a structured query language for the SPO model that provides familiar SQL - like declarative syntax that is easier for the programmers. The current work implemented SPOQL language over the SP-Algebra.

Parsing SPOQL queries is a more involved task than parsing SQL queries, due to the fact that important query equivalences do not hold in SP-Algebra. As a result, some query translations are incompatible. *Eager evaluation* is used for parsing SPOQL queries. According to *eager evaluation*, all operations are applied in the defined order of precedence as soon as they can be executed. The new SPDBMS server with SPOQL can replace the old SPDBMS server and application using SPDBMS as the back end for storage of probabilistic data. This can be used in the system being developed for Welfare-to-Work modeling project [20]. The current work also extends SP-Algebra with a new mix operation and enhances selection by comparing context variables of SPOs.

Bibliography

[1] D. Barbar'a H Garcia-Molina, and D.Porter. The management of probabilistic data. *IEEE Trans. On Knowledge and Data Engineering, 4:487-502,1992*

[2] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proc.VLDB'87*, pages 71-81,1987.

[3] Alex Dekhtyar, Raphael Finkel, Judy Goldsmith, Beth Goldstein, and Cynthia Isenhour. Adaptive decision support for planning under hard and soft constraints. In *Proceedings of the AAAI Spring Symposium on Challenges to Decision Support in a Changing World,*pages 17-22, Stanford University, Palo Alto,CA,2005

[4] Alex Dekhtyar, Judy Goldsmith, and Sean R. Hawkes. Semistructured probabilistic databases. In *Proc. Statistical and Scientific Databases Management Systems*,pages 36-45,2001.

[5] Alex Dekhtyar, Judy Goldsmith, Huaizhi Li, and Brett Young. Bayesian advisor Project I: Modelling academic advising. Technical Report TR 323-01, University of Kentucky, March 2001.

[6] D.Dey and S. Sarkar. A probabilistic relational model and algebra, *ACM Transactions on Database Systems*, 21(3):339-369, 1996.

[7] Debabrata Dey and Sumit Sarkar. PSQL: a query language for probabilistic relational data. Data Knowl. Eng.,28(1):107-120,1998.

[8] T. Eiter, J. Lu,T. Lukasiwicz, and V.S. Subramanian. Probabilistic object bases. ACM Transactions on Database Systems, 26(3):264-312, 2001.

[9] Edward Hung, Lise Getoor,and V.S. Subrahamanian. Probabilistic interval xml. In *ICDT,* pages 361-377,2003

[10] Edward Hung,Lise Getoor, and V.S. Subrahmanian. Pxml: A probabilistic semistructured data model and algebra. In *ICDE*, 2003

[11] E Kornatzky and S.E. Shimony. A probabilistic object data model. *Data and Knowledge Engineering,* 12:143-166, 1994.

[12] V.S. Lakshmanan, N. Leone,R. Ross, and V.S.Subrahmanian. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419-469,1997.

[13] Kevin Krol Mathias, Cynthia Isenhour, Alex Dekhtyar, Judy Goldsmith, and Beth Goldstein. Eliciting and combining influence diagrams: Tying many bowties

together Technical Report TR 453-06, University of Kentucky, March 2006.

[14] Andrew Nierman and H.V. Jagadish, ProTDB:Probabilistic data in XML. In *Proceedings of the28th VLDB Conference*,2002.

[15] Judea Pearl. *Probablistic Reasoning in Intelligent Systems:Network of Plausible Inference*. Morgan Kaufmann,1988.

[16] S. Prabhakar R. Cheng, D. Kalasniov, Evaluating probabilistic queries over Imprecise data. In *Proceedings of the ACM SIGMOD,*2003.

[17] S. Prabhakar R. Cheng, Sarvjeet Singh. U-dbms: A database systm for managing Constantly-evolving data. In *Proceedings of the 31st VLDB conference,2005*.

[18] Robert Ross, V.S. Subrahamanian, and John Grant. Aggregate operators in Probabilistic data. *Journal of Intelligent Information Systems,*25(3):293-332,2004.

[19] Wenzhong Zhao, *Probabilistic database and their applications.* PhD thesis, University of Kentucky,2004.

[20] Wenzhong zhao,Alex Dekhtyar,Judy Goldsmith. A Framework for Management of semistructured probabilistic data *Journal of Intelligent Information Systems* Vol 25,No 3,pp.293-332

[21] Wenzhong Zhao, Alex Dekhtyar, Judy Goldsmith, Erik Jessup, and Jiangyu Li. Building bayes nets with semistructured probabilistic dbms.

[22] Steven John Metsker, *Building Parsers with Java.*

SPOQL Grammar

< spoqlquery > :: = **SELECT** < *selectlist* >
FROM < *spolist* >
[**WHERE** < *selectioncond* >]
[**CONDITIONAL** < *conditionlist* >

*<selectlist> := * | < projectioncond >*
<projectioncond> := < AtomicProjectionList > |
< projectioncond >,< projectioncond >

<AtomicProjectionList> is defined in Definition 2 of Chapter 2

<spolist> :=< sp-relation > |
< sp-relation > < combOp > < sp-relation >

<sp-relation> := < Name > |
(< spoqlquery >)< Name > |
(< spoilst>)< Name >

<combOp> := **','|'JOIN'|'TIMES'**

<selectioncond> := <AtomicSelectioncond> |
[(]< selectioncond > **AND** *< selectioncond > [)] |*
[(]< selectioncond > **OR** *< selectioncond > [)]*

<AtomicSelectioncond> is described in Table 2.1 of Chapter 2

<conditionlist> := < AtomicCondExp > |
< conditionlist > **AND** *< conditionlist >*

<AtomicCondExp> is defined in Definition 3 of Chapter 2

Vita

Date of Birth:	July 25, 1982
Place of Birth:	Tirupati, India
Education:	Bachelor of Technology in Computer Science & Engineering Sri Venkateswara University, Tirupati, India, 2003

Professional Publications

"Structured Queries for Semistructured Probabilistic Data" in TDM 2006, pages 11-18, 2006.

Praveen Gutti