University of Twente
Enschede - The Netherlands

Department of Electrical Engineering, Mathematics and Computer Science
Database group

Managing Continuous Uncertain Data by a Probabilistic XML Database Management System

Theodoor Scholte

M.Sc. Thesis
28 May 2008

Graduating committee:
Dr. ir. Ander de Keijzer
Dr. ir. Maurice van Keulen
ir. Sander Evers

"Doubt is not a pleasant condition, but certainty is absurd." –Voltaire

Abstract

Database systems are widely used in today's world. Almost every information system contains one or more databases. From a traditional perspective, databases are used to store precise values about objects in the 'real world'. However, many information is uncertain or imprecise. Consider, for example, sensor applications. Sensors produce uncertain and imprecise data since readings of sensors are inherently imprecise and uncertain. Current database management systems are not able to store, manipulate or query continuous uncertain data unless through user-defined attributes. However, this approach delegates the responsibility of managing the uncertainty associated with the data to the end-user.

In many situations, the uncertainty associated with the data is distributed continuously, the data can be represented in terms of a continuous probability distribution. In this thesis, we present an extension to an existing probabilistic data model, resulting in a data model which is capable of storing continuous uncertain data in XML documents. We give a sound semantical foundation to this data model. The probabilistic XML data model is based on the probabilistic tree. In the probabilistic tree, elements and subtrees can be associated with probabilities. Our extension to the probabilistic XML data model extends the probabilistic XML data model in such a way that probability density functions can be associated with elements.

Instead of enumerating explicitly the data values with their associated probabilities, a probability density function represents a continuous probability distribution in terms of integrals, it can represent the probability that an element attains a value on a specific interval. In order to query this data, we present a query language containing query operations that are based on probability theory. We show how querying of continuous uncertain data works using a sound semantical foundation. Next, we introduce some new query operators supporting the aggregation of continuous probability distributions using the same semantical foundation. An aggregation operator accepts a number of histograms representing continuous probability distributions, aggregates them and returns one histogram representing a continuous probability distribution.

A proof of concept demonstrates the outcomes of our study towards the management of continuous uncertain data. This proof of concept allows the end-user to query XML documents containing continuous uncertain data.

Dankwoord

Na 7 jaren studeren rond ik mijn studie Informatica af door het doen van een onderzoek. Deze scriptie is daarvan het resultaat. Het afgelopen halfjaar heb ik onderzoek gedaan naar het beheren van continue onzekere data. En ik moet zeggen dat het leek alsof ik in deze periode weer was teruggekeerd naar de peutertijd, spelend met de blokkendoos. In je peutertijd bouw je met blokken huizen en gebouwen en als het resultaat niet voldoet, dan sloop je het en begin je weer opnieuw. Zo gaat het ook met onderzoek doen, je stelt een hypothese op, je werkt naar een theorie, je ontwikkelt een model. Door middel van een prototype kan je experimenten uitvoeren waardoor je weet of je aanpak werkt. Werkt de aanpak niet, dan gooi je het weg en begin je weer opnieuw. Net als dat ik het spelen met blokken leuk vond, vind ik onderzoek doen ook hartstikke leuk. Het was een leuke afsluiting van een fantastische tijd die ik hier op de Universiteit Twente heb gehad.

Onderzoek doen heeft ook mindere kanten. Het is een nogal solitaire bezigheid waardoor het saai kan worden. Dan zijn er een aantal oplossingen. 1. Je maakt een wandeling over de prachtige campus. 2. Je gaat met je begeleiders of andere mensen op de vakgroep praten. 3. Je gaat vroeg naar huis. Regelmatig had ik een paar uur ”bedenktijd” nodig voordat ik überhaupt een eerste zin op ”papier” kon zetten. Een wandeling maken, het bekende DB-rondje, bood dan uitkomst. Maar gelukkig heb ik ook veel aan mijn begeleiders en de mensen op de vakgroep gehad om inspiratie en nieuwe ideeën op te doen. Ook de zogenoemde DB Colloquia hebben daar zeker aan bijgedragen.

Deze scriptie was niet tot stand gekomen zonder de hulp van heel veel mensen. Allereerst wil ik mijn begeleiders, Ander de Keijzer en Maurice van Keulen, bedanken. Zij waren het die het initiatief namen om onderzoek te gaan doen naar het beheren van continue onzekere data. Ik heb niet alleen hun data model voor discrete onzekere data kunnen gebruiken, maar ze hebben me ook enorm geholpen met het uitbreiden van dit model zodat het nu ondersteuning biedt voor continue onzekere data. Hun enthousiasme gaf mij vertrouwen om door te gaan met het onderzoeksproject, ook als het even wat minder ging. Daarnaast hebben ze altijd het geduld gehad om naar mij te luisteren ook al had ik soms moeite om datgene wat ik wilde zeggen, juist te formuleren. Ook zal ik de voortgangsbesprekingen die we hadden niet snel vergeten. We hadden deze besprekingen niet vaak, maar ze mondden vaak wel uit in interessante, heftige en diepgaande discussies over de inhoud. Mijn derde begeleider, Sander Evers, heeft me enorm geholpen om het verslag leesbaarder te maken voor de buitenstaander en wist me nog op een aantal onjuistheden te wijzen. Uiteraard, ben en blijf ik, verantwoordelijk voor de overgebleven onjuistheden.
Tijdens mijn onderzoek heb ik een prototype gemaakt wat op een experimenteel database management systeem draait. Dit prototype was niet tot stand gekomen zonder de hulp van

Jan Flokstra. Ik kon hem op de meest rare tijden nog e-mailen en dan loste hij een probleem snel voor me op.

Uiteraard kan ik mijn ouders niet vergeten te bedanken want zonder hen was ik nooit zover gekomen. Inhoudelijk hebben zij natuurlijk weinig kunnen bijdragen aan deze scriptie, maar zij hebben wel mijn eindeloze verhalen over de voortgang aangehoord. Daarnaast, zijn zij het, die het überhaupt mogelijk hebben gemaakt om te studeren, zonder al te veel druk er achter te zetten. Ik heb daardoor kunnen genieten van mijn studie en alle nevenactiviteiten, zonder financiële zorgen hoeven te maken, iets wat in mijn ogen erg belangrijk is.

Theodoor Scholte
Enschede, 28 mei 2008

Contents

Chapter 1

Introduction

Database Management Systems (DBMSs) play an important role in today's world. Almost every information system contains one or more databases [5]. The application domain ranges from those things that we daily use (e.g. phone book on a cellular phone) to highly sophisticated systems (e.g. container terminal management system in a main port). From a traditional perspective, databases are used to store precise values about objects in the 'real world'. Though, today, there are many applications in which data is uncertain and/or imprecise. It is frequently the case that these applications use a conventional DBMS and process the uncertainty outside the DBMS.

1.1 Motivation

More than one decade ago, researchers addressed the importance of the problem of managing uncertain data by a Database Management System [21]. Although, considerable research effort has been conducted to solve this issue, the management of uncertain data by a Database Management System is still an active research topic. The reason for this is that there is a wide variety of different models supporting the storage of uncertain data, each with its own features. Similarly, there is a broad range of applications each having its own specific requirements. In the next few sections we will mention several application domains of uncertain data.

1.1.1 Sensors

> *"Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives." [24]*

Ubiquitous computing is a paradigm which breaks the traditional borders in the interface between person and computer. The computer will be integrated into everyday's life. Applications using the ubiquitous computing paradigm are emerging. In many ubiquitous systems sensors are one of the core components. For example, due to location sensors, a car knows where it is. In case of an accident the car is able to notify the rescue operators with a cell phone.

Another example is the adaptable room conditions in an office building. A user wears clothing with an RFID-tag. When entering the room, the temperature control system 'recognizes' the user and its preferences. Sensors of the control system are measuring the temperature and humidity in the room at regular intervals, and transmit the data to a central processing component. The control system adjusts the temperature and humidity in the room according to the user's preferences.

Sensors may produce unreliable data due to missed readings and/or erroneous or imprecise values. Another problem occurring in applications with sensors and databases is that there is a gap in time between the actual value in the 'real world' and the database value [10]. Then, queries will use the old database value instead of the accurate value in the 'real world'. One way to solve these problems, is to store imprecise data along with probabilities that indicate the confidence that an imprecise value stored in the database is the accurate value in the 'real world'.

1.1.2 Science

In experimental scientific research like earth-sciences, psychology, chemistry and biology, scientists conduct a lot of experiments that produce raw data which should be stored and processed. This data is often imprecise and subject to uncertainty and errors that cannot be modeled by traditional data management systems [25]. The causes of uncertainty and errors range from reading errors, errors in filled questionnaire forms, transmission noise to other errors in unreliable information sources or unreliable systems. Experimental data can be transformed or processed further using certain operations (like aggregation). Subsequent experiments can be carried out using the derived data resulting in new data. This data may alter the previously approximated data or it might change the confidence scores of it. The values that will be stored, might be inexact or uncertain. In many situations, it is desirable to store the data along with probabilities or with confidence values. This motivates the need for a data management system with uncertain capabilities.

1.1.3 Analysis and Forecasting

Consider economic analysis and forecasting over a certain period of time. In general, the following holds: the more information becomes available, the better becomes the prediction [11]. It is basically the same as with the weather forecast. The forecast for the next day is much more accurate and certain than the forecast for three days in the future. Most macroeconomic data should be treated as uncertain data; they are estimates rather than measures. This is because errors arise due to the fact that the data is often based on small samples. The estimates are revised as more information has become available and statistical methods have been reviewed. The data published in the past can be seen as probabilistic data. Using this data together with historical experience can improve the way of estimating the confidence of earlier received macroeconomic data. Closely related to this area are applications which analyze data about individuals to create elaborate profiles [25]. Typical examples include: of risk assessment (credit and insurance) and targeted advertising. Another closely related topic is Online Analytical Processing.

1.1.4 Online Analytical Processing

Business Intelligence is increasingly being used to support strategic decision-making. Business Intelligence applications like Online Analytical Processing are used to gain knowledge in a certain business domain resulting in competitive advantages. OLAP applications are used by managers supporting them in making decisions. OLAP queries frequently involve a significant part of the database, they are complex and updates on the data are infrequent. Data warehouses are special databases used by OLAP applications. Data warehouses are very large, have schemas which differ from schemas of "normal" online transaction processing DBMSs (OLTP systems) and the data itself is different from what is normal in OLTP systems. In practice, warehouses contain uncertain data because data has been obtained from multiple data sources. When integrating data from different sources, each having its own schema, conflicts may occur. Moreover, the input data does not only arrive from the OLTP systems of an enterprise [5]. Think, for instance, about a customer of a company filling out a questionnaire form on a website and the customer enters a wrong city name. Typically, OLAP applications deal with problems due to space limitations of the warehouse. Sometimes, it is impossible to store all the data of the business in the warehouse. One solution then is to perform OLAP queries on smaller samples that have been extracted from the large data set. However, this simplification introduces imprecision [16]. Another example of uncertainty in OLAP applications is forecasting [15]. Data warehouses are used to maintain a history of forecasts about, for instance, sales. These forecasts are inherently uncertain.

1.1.5 Data Integration

Consider a person who has address books on his cellular phone, PDA and desktop computer. All these information sources contain information about persons. The person would like to have one information source to find an address. Data integration can assist the user because it can present the various information sources as one single source. The data in different information sources can refer to the same 'real world' object or to multiple objects. If the data is about the same object, data can be conflicting. For instance, the phone number of the same person is different. Then there are several possibilities: a person can have two different phone numbers or only one of the phone numbers is valid. Data can be conflicting for several reasons: a miss-typed phone number, an old phone number etcetera. We can choose to use one phone number out of the different possibilities, but this may lead to incorrect results. A better approach when integrating information sources is to store for all conflicting values the possibilities with their associated probabilities [23]. The probabilities are a measure of truth that a certain possibility indeed occurs in the 'real world'. The result of the integration process is an integrated information source containing uncertain data. A probabilistic database management system is required to store the result. The main benefit of this approach is the unattended way of integration [12].

1.1.6 Approximate Queries

There are scenarios in which huge amounts of data are generated. Due to limited resources (e.g. bandwidth, disk space, computing power), it is often infeasible to store all the exact values [10, 25]. In some scenarios when querying the database, the end-user is more interested in an estimated result with a lower execution time than in an exact result taking ages to

compute. One approach is to produce approximate answers using a small data set or statical tools. Another approach is to store the approximate values along with a degree of uncertainty; a DBMS supporting the management of uncertain data is required then.

1.2 Problem statement

The previous section illustrates some application domains in which (continuous) uncertain data is involved. Many applications in these domains make use of database management systems to maintain and manipulate data. The management of (continuous) uncertain data by traditional database management systems is hard to achieve. This is because those systems assume that data is certain, precise and complete. As shown in the previous sections, information about the real world can be inherently uncertain, imprecise or incomplete. The management of continuous uncertain data by current database management systems is only possible through user-defined attributes. Then, the database management system delegates the responsibility of managing continuous uncertain data to the end-user. Moreover, there are currently no database management systems available which are capable of managing continuous uncertain data using a semi-structured data model. This leads us to the main problem which this thesis will focus on:

> *Designing a* semi-structured *probabilistic database management system which is responsible for maintaining and manipulating* continuous uncertain data *in such way that probabilistic query operators are able to produce* imprecise, but correct, *answers.*

To clarify the problem statement, we elaborate three terms:

- Semi-structured
  Data can be represented in different data models, examples are the relational and the semi-structured data models. Discrete uncertain data can be represented in a semi-structured data model (XML) in a more natural way than in the relational model. In order to fully support the representation of uncertain data in XML, the representation of *continuous* uncertain data should be examined as well. We are also interested in using XML for the representation of continuous uncertain data since XML is used more and more for storing data and it is the *de facto* standard for data exchange between organisations.

- Continuous uncertain data
  With the term "continuous uncertain data", we mean data that can be denoted as stochastic variables each having a univariate continuous probability distribution.

- Imprecise, but correct
  Data which describes a certain aspect of the real-world can be represented by one value, a set of values or an interval. A probability distribution can be added to the representation to indicate the likelihood that a value reflects the actual state of the real world. In case the value of an attribute is set- or interval-based, a query for that attribute should return (a part of) the corresponding set or interval which is imprecise, but correct.

1.3 Research questions

In the previous section we identified the main problem this thesis will focus on. This problem can be solved by answering several research questions:

1. *Which additions to existing data models are necessary in order to support the representation of continuous uncertain data in XML?*
   In order to integrate continuous uncertainty in a DBMS and making the system responsible for maintaining and manipulating continuous uncertain data, a data model is required. The development of such model is also the main research challenge that has to be addressed in this thesis.

2. *Which semantic foundation is suitable for storing continuous uncertain data?*
   The data model we will introduce in this thesis has to be complete and closed. The data model has to deal with data which is associated with continuous probability distribution in a correct way. For these reasons, the model needs a (semantic) foundation.

3. *How can we provide support for querying continuous uncertain data?*
   Besides the development of a data model supporting the representation of continuous uncertain data in XML, we need query operations that allow the end-user to retrieve information from continuous uncertain data.

   How should the semantic foundation be applied in order to support querying continuous uncertain data in an intuitive way?
   The query operations which we will provide should be intuitive to use. Next, the results generated by the query operations, should be intuitive to interpret (understand).

   Which query operations are useful when querying continuous uncertain data?
   At this moment, we have no idea which query operations are useful for the end-user. We will examine which operations *could* be useful.

4. *How can the theoretical concepts as a result from the research on the management of continuous uncertain data be practically applied?*
   The answers on the previous research questions may contain several new concepts. We are interested if these concepts are applicable in a feasible way. This question will be answered by developing a proof of concept.

5. *What is the behavior of a probabilistic XML DBMS supporting the management of continuous uncertain data in terms of efficiency and accuracy of query answers?*
   In database research, benchmarks are used to evaluate the performance of database management systems. By developing the proof of concept, we build new elements that will run on top of an existing XML DBMS. Since we add new mechanisms to a DBMS, we are interested in their behavior. We will examine whether the approach that has been taken is feasible in terms of efficiency and accuracy of query answers.

1.4 Scope

The main goal of our research is to provide support for the management of continuous stochastic variables in an XML database management system. In order to meet this goal, we determine several subgoals:

- The development of a data model supporting the storage of continuous probability distributions in semi-structured data.
- Extending a query language with operations that allow users to query on semi-structured documents containing continuous stochastic variables.
- Developing a prototype in order to prove the concepts.

Besides these goals, there are also some goals or activities that are related to the topic of this thesis but are out of the scope of this thesis. We will summarize them here.

- Our work is inherently related to mathematics and especially statistics and probability theory. In this thesis, we will propose query operators that are capable of querying continuous distributions resulting in probabilistic answers. Query operators include the probability, the expected value and variance of a continuous distribution and so on. The probabilistic answers can be computed in two ways.

  Exact answers
  Standard continuous distributions like gaussian, gamma, beta and uniform have well-known probability density functions. One approach to compute an exact answer is based on probability theory, answers are computed by taking the integral of the probability density function. Another approach is to use a distribution-specific function, with the parameters of a standard continuous distribution as input, to compute the answer. An example is a query for the expected value of a continuous uniform distribution, the expected value can be computed with the parameters a and b using the following distribution-specific function: $\frac{a+b}{2}$.

  Approximations by histograms
  A continuous distributions can be approximated by a histogram. Histograms are capable of approximating non-standard continuous distributions. A continuous distribution is called "non-standard" if the probability density function is not a trivial one, e.g. the probability density function is really hard to compute or it is unknown. Query operators that operate on histograms will return approximated answers.
  Our goal is to provide formal definitions for the data model and the query language. The query language should be based on probability theory and the operators are defined as if they return exact and correct answers, no approximations. However, it is not required that the proof of concept produces exact answers or formulas when these are too complex or too time-consuming to compute.

- Performance and efficiency are currently not the most important issues. We would like to focus on the concepts and proof them using a prototype. Query optimization, indexing and overall database performance are out of the scope of this thesis.
- The prototype will be build to demonstrate the concepts that are introduced in this thesis (proof of concept). For that reason, we are currently not interested in usability aspects of the prototype.

1.5 Approach

The approach taken in this research is that of the constructive research. In this thesis, we present new theoretical knowledge by means of a data model and a query language. This theoretical knowledge can be seen as a solution to the problem of managing continuous uncertain data by a semi-structured DBMS. The solution is based on many sources that contain theoretical information about uncertain data, e.g. papers and so on. In order to proof the concepts that are presented in the theoretical knowledge, we will develop a prototype and perform some tests or benchmarks on it. This proof of concept forms also an important part of the validation. The proof of concept shows us, for example, whether the data model allows the modeling of continuous uncertain data in a sufficient way; it will give us feedback on the shortcomings of the data model and the query language. The tests or benchmarks allow us to make conclusions about whether the practical use of the data model and query language is feasible in terms of the quality of query answers and query response time.

1.6 Outline

One of the first activities that will be carried out is the extension of an existing data model [23]. This data model is based on the possible world semantics [27] and it has already support for the representation of discrete stochastic variables. We will extend this model in chapter 3 in such a way that it supports the representation of continuous uncertain data. Next, in chapter 4 a query language is proposed. This query language includes constructs for querying uncertain data modeled in the proposed data model. The development of the data model and a corresponding query language results in new concepts. In order to proof those concepts, we will develop a prototype that will be described in chapter 5. The prototype will be evaluated in chapter 6 using benchmarks. Chapter 7 includes the conclusions and future work. But, first, we start with discussing the related research on the management of (continuous) uncertain data in chapter 2.

Chapter 2

Related work

In this section we will discuss related work on uncertain data management. Different data models such as the relational model, semi-structured models, probabilistic and possibility models will be discussed. Moreover, existing research projects will be mentioned. Besides explaining different data models, we will explain general concepts such as what imperfect information is and what possible world semantics is. An important point in this chapter is the discussion of related work on continuous uncertainty.

2.1 Imprecise, uncertain, and erroneous data

The title of this section mentions some terminology used in previous work on uncertain data management. Although, terms like *imprecise*, *uncertain* and *erroneous* might suggest that their meaning are all the same, this is certainly not the case. A. Motro gave a good interpretation [17] to these terms. In this section we will give a short summary.

Erroneous information is information that is different from true information. This type of imperfect information influences the integrity of information systems. One kind of erroneous information is inconsistent information. Different representations of a particular part of the real world that are conflicting can be considered as inconsistent information. An example is the birth date of the author which is stored in a database as August 4th, 1982 but which is in fact August 3th, 1983.

Another kind of imperfect information is imprecise information. Imprecise information is represented as a set of alternatives and the real value is one of those alternatives, so the real actual value is not known a-priori. Consider the temperature of a room stored in a database as a range between 18 and 20 degrees, while the real room temperature is 19 degrees. There are two extreme kinds of imprecise information: precise values and incompleteness. A precise value means that the set of possible alternatives contains exactly one single alternative. Incompleteness means that no information is available. Note that the representation then indicates that there is no information available.

Uncertain information expresses doubt about if our knowledge about the real world is correct. The doubt is quantified with confidence values. These confidence values can be associated with each representation of the real world in an information system. An example is John Doe who has phone number 1111 with probability 0.5 and phone number 2222 with probability 0.5.

2.2 Possible world semantics

The data stored in a database describes some part of the real world. In situations where available information is certain and complete, the similarity between the database and the real world is clear. However, when the database contains data that is uncertain, the database represents a collection of possible appearances of the real world. Those appearances are also called states or *possible worlds*. A possible world describes a set of objects in the real world. An object in the real world can have different (or multiple) interpretations. Then, different information about that object exist. Another way of describing this is to say: "there exist several *possibilities* or *alternatives* for an object in the real world". A possible world is constructed by choosing one alternative among a collection of representations for each of the real world objects in the database. If one of the possibilities represents that the real world object does not exist, this should also be considered and treated as if it is an alternative. The possible world semantics are especially helpful for defining the mechanisms used for querying and manipulating data in uncertain databases. As we will show in one of the following chapters, evaluating a query on an uncertain database is equivalent with evaluating the query over each individual possible world and collecting the results that originate from each possible world. In this thesis we interpret information represented in the database under the Closed World Assumption stating that all the information not explicitly represented in the database is supposed to be false. This is contrary to the Open World Assumption in which negative information must be explicitly represented in the database. The possible world semantics is such a fundamental concept in research on management of uncertain data that most of the related projects are based on this theory.

2.3 Uncertainty in relational databases

Uncertain data can be encoded in conventional relational tables. The data model of most existing uncertain data management systems, MystiQ [8], Orion [10, 22], PrDB [20], ProbView [14] and Trio [4, 18], extends the relational data model. Those data models support one or more of the following concepts:

- Type-1 probabilistic relations
  Type-1 uncertainty refers to confidence if a tuple belongs to a relation or not. Consider table 2.1(a). The table represents a part of my personal address book. It is not really likely that my address book contains the phone number of the Dutch Queen, where it is very likely that the address book contains the phone number of one of my fellow students, Ruud van Kessel.

- Type-2 probabilistic relations
  With Type-2 uncertainty, the value of the key-attribute is deterministic but values of other attributes in the relation may be uncertain. Table 2.1(b) shows a relation which depicts where Kings and Queens of different countries around the world live. There is no uncertainty about the country where the King or Queen lives. Since the attribute-values of the field "town" for Queen Beatrix and King Carl XVI Gustaf represents uncertainty, it is not possible to tell in which village they live with complete certainty, based on this list.

Table 2.1: Examples of different concepts related to managing uncertain data.

(a) Type-1 probabilistic relations

Name	**Phone number**	**Probability**
Beatrix van Oranje	+31 70 1234567	0.01
Ruud van Kessel	+31 6 12345678	0.99

(b) Type-2 probabilistic relations

Name	**Country**	**Town**
Beatrix van Oranje	The Netherlands	The Hague/0.9 Amsterdam/0.1
Carl XVI Gustaf	Sweden	Stockholm/0.5 Malmo/0.5

(c) Null values

Name	**E-mail address**
Sander Evers	null
Ander de Keijzer	a.dekeijzer@utwente.nl
Maurice van Keulen	keulen@ewi.utwente.nl

(d) Tuple alternatives

Name	**Country**	**City**	**Probability**
Teade Scholte	The Netherlands	Ede	0.5
Teade Scholte	France	Condorcet	0.5
Theodoor Scholte	The Netherlands	Enschede	0.9
Theodoor Scholte	France	Biot	0.1

- Incompleteness
  The value of an attribute in a relation is sometimes unknown; the information is incomplete. This can be expressed by a NULL value. A NULL value models the lack of information about a value explicitly. Table 2.1(c) represents another part of my address book. The table lists the names and e-mail addresses of my supervisors. The e-mail address of supervisor ”Sander Evers” is currently not known, this has been indicated in the table with ”null”.

- Tuple alternatives
  The presence of tuple alternatives in a database means that there are several possible tuples for one tuple available in the database. Each possible tuple can be associated with a probability indicating the likelihood that the possible tuple represents some part of the real world. Table 2.1(d) shows a list of places where my parents live (and where I live). There are two alternatives in case of my parents, each having a probability.

- Mutual exclusiveness
  A database, supporting the modeling of mutual exclusiveness, is able to manage the relation between data items that represent propositions. These propositions cannot be true at the same time. An example is 'Theodoor saw Ander OR Theodoor saw Maurice'. Note that Type-2 probabilistic relations model also some kind of mutual exclusiveness. However, with Type-2 probabilistic relations, a probability is associated with each alternative.

Many uncertain data management systems are built on top of a traditional RDBMS extended with (stored) procedures, functions and data types implementing the uncertainty features. Most uncertain data management systems provide support for querying uncertain data by accepting a modified version of SQL. In general, the modified language has special constructs for querying uncertain data.

2.4 Uncertainty in semi-structured data

Semi-structured data models have also been used as a model to represent uncertain data. In [12] two strategies were identified for modeling uncertainty in semi-structured data models: *event based* and *choice point based*. With the first one, a choice for a certain alternative in a tree is based on a pre-specified event occurring. This choice invalidates the other possible alternatives that are present in the tree. Each event has an associated probability. The choice point based strategy is more dependent on the *structure* of the tree. The strategy requires that at specific points in the tree, a decision has to be made between children, each having an associated probability. The result of such approach is an entire subtree representing a path or trace of decisions made. This subtree invalidates the other possible alternatives.

Hung et al. [13] proposed the PXML data model. It is a complex data model supporting the probabilistic representation of arbitrary distributions over relationships between objects and arbitrary distributions over the object's value. In this data model, a probabilistic instance does not have to be tree structured. A probabilistic instance has to be acyclic which means that one node can have two different parents, the only restriction is that cycles are not allowed. This data model is event based. The same holds for the Fuzzy tree model of Abiteboul and Senellart [1]. Their model allows conditional reasoning and modeling of complex dependencies between events. ProTDB of Nierman et al [19] and the Probabilistic XML data model proposed in [23] are both choice point based probabilistic models. Each node is not treated as an independent fact but its probability is dependent on the probabilities of its ancestors (except the root). In both models, a special type of node expresses possibilities (alternatives) and each possibility is associated with a node indicating the probability. However, the model proposed in [23] handles mutual exclusiveness in a more natural way because it does not require that mutual exclusive possibilities have to be specified explicitly.

2.5 Continuous uncertainty

Many applications require the management of continuous uncertain data at the database level. Sensor data, scientific data and geographical information are all examples of data containing continuous uncertainty. Continuous uncertain data can be represented as a continuous probability density function (PDF) like Gaussian or the Gamma function with some

Table 2.2: A small address book

Name	**Phone number**
John	1111
John	2222
Peter	3333

parameters. The integral over such functions on a particular interval results in the probability that the value of the continuous stochastic variable lies on that particular interval. Continuous uncertainty can also be approximated by a histogram, hence the continuous distribution has to be sampled. At Purdue University, a probabilistic model was developed in the Orion project by Singh et al. [10, 22]. It supports the management of discrete as well as continuous uncertain data using the *relational* data model. The model is able to represent continuous distributions very efficiently and handle them accurately by using the symbolic form of a continuous distribution. Discrete approximations (e.g. histograms) are not necessarily required. However, the model uses discrete approximations for the representation of non-standard distributions. Moreover, the model supports both Type-1 and Type-2 uncertainty and it can handle correlations within and across tuples. The prototype has been developed as an extension of PostgreSQL. Orion is currently the only project investigating the management of continuous uncertain data as far as we know.

2.6 Querying uncertain data

As mentioned earlier, a database supporting the management of uncertain data, represents a set of possible worlds. Following the possible world theory, querying uncertain data means that the query is evaluated in every possible world. The results from each of the worlds are collected by the database management system and the union of the results is presented to the user as an answer to the query that was posed by the user. Consider an address book as depicted in table 2.2. The database contains two tuples, one tuple has two alternatives. The total number of possible worlds equals $1 * 2 = 2$. A query for the phone number of John is evaluated in each possible world. Collecting the results from each of the worlds and applying the union operator over the results would yield the following result: $\{1111, 2222\}$. As we will see later, a database which is storing continuous uncertain data, holds theoretically an infinite number of possible worlds. Querying this kind of data using the possible world semantics, means from a theoretical perspective that the query is evaluated in an infinite number of possible worlds.

The uncertain data management systems developed in the Trio project at Stanford University and in the Orion project at Purdue University, are built on top of an existing database management system. The SQL query language is extended with functionality supporting query evaluation over uncertain data. Querying uncertain data leads to query answers containing uncertainty. Then, the question arises how the quality of the answer can be measured. Ander de Keijzer and Maurice van Keulen contributed to this subject by proposing redefined functions for precision and recall [12]. Precision and recall are measures for the quality of an answer indicating the exactness and completeness, respectively. The measures were adapted in such a way that probabilities are taken into account.

Table 2.3: A part of a database containing advertisements of real estate objects.

AdId	**AgencyId**	**City**	**Type**	**Rooms**	**Surface**	**Furnished**	**Price**
101	303	Amsterdam	Apartment	2	?	yes	800
102	445	Amstelveen	Apartment	3	?	no	650
103	303	Hilversum	Apartment	2	45	?	600

Table 2.4: Factors describing the correlations and probabilities among tuples and attributes

AdID	$f(t.E)$
101	0.2
102	0.8
103	0.6

t_{103} Furnished	$f(t_{103}.Furnished)$
yes	0.5
no	0.5

City	**Type**	**Rooms**	**Surface**	**Price**	$f(city, type, rooms, surface, price)$
Amsterdam	Apartment	25	2	800	0.25
Amsterdam	Apartment	30	2	800	0.5
Amsterdam	Apartment	35	2	800	0.25
Amstelveen	Apartment	30	2	650	0.1
Amstelveen	Apartment	40	2	650	0.8
Amstelveen	Apartment	50	2	650	0.1

Table 2.5: An example of tuple fields having set-values.

ID	**Name**	**City**	**Phone**
1	John Doe	{Eindhoven, Groningen, Maastricht}	{0845533228, 0622222222}
2	Peter Pan	{Amsterdam, Enschede, Utrecht}	{0843335599, 0611111111}

2.7 Lineage

A database having support for lineage means that it has support for modeling the trace to the origin of data residing in a tuple. Lineage identifies the derivation of a tuple to other tuples. These tuples can be inside the database (internal lineage) or in an external data source (external lineage). Lineage is closely related to uncertainty because it is a powerful mechanism to trace where uncertainty is originating from. The mechanism allows the correlation and coordination of uncertainty between query results and data residing in base tuples (origin). Trio of Widom et al. [4, 18] is an example of a relational database management system that has support for representing and manipulating uncertainty and lineage.

Table 2.6: An example of world-set decomposition.

$t_1.Name$
John Doe

$\times$

$t_1.City$
Eindhoven
Groningen
Maastricht

$\times$

$t_1.Phone$
0845533228
0622222222

$\times$

$t_2.Name$
Peter Pan

$\times$

$t_2.City$
Amsterdam
Enschede
Utrecht

$\times$

$t_2.Phone$
0843335599
0611111111

2.8 Probabilistic dependencies

Probabilistic dependencies are related to the topic of lineage. In the probabilistic models that are based on the relational data model, one can distinguish intra-tuple and inter-tuple correlations. Inter-tuple correlations refer to dependencies between tuples. Historical dependencies between tuples can be used to represent the result of prior database operations on base tuples. In other probabilistic models, lineage is used for this purpose. Intra-tuple correlations capture the dependencies between attributes inside a tuple. In semi-structured probabilistic models, dependencies or correlations can be expressed in a natural way by modeling child nodes that depend on their ancestors. An interesting project to mention is PrDB. PrDB is a probabilistic data model proposed by Getoor et al. [20] that can express both Type-1 and Type-2 uncertainty. One of the remarkable features is that it supports so-called shared correlation structures. This mechanism prevents that probability functions and correlations have to be copied many times when probabilities do not change from tuple to tuple. The idea of the correlation structure is based on a parameterized factor representing uncertainty and correlations over parameterized variables which involve attributes or tuples (existence). Consider table 2.3, it represents a database containing information regarding real estate objects that are for sale or for rent. Unknown attribute values are shown as question marks. The corresponding probabilistic database is depicted in table 2.4. Table 2.4 shows the parameterized factor $f(t.E)$, this factor describes the likelihood of each advertisement existing. The table also shows the parameterized factor $f(t_1 03.Furnished)$, this factor describes the probability that the apartment described by tuple 103 is furnished or not. The surface of the apartments, that are described by advertisements with ID 101 and 102, is unknown. The parameterized factor $f(city, type, rooms, surface, price)$ describes the likelihood for the combinations of (city, type, rooms, surface, price). Due to the data sharing features, the model is space efficient. Another interesting project is MayBMS [2, 3], the model used by MayBMS allows world-set-decompositions. The concept is based on the decomposition of world sets into several relations such that the product of the relations represent a possible world. Table 2.5 shows an example of a database having set-based attribute-values. In table 2.6, we see that the world-set relations are decomposed into several relations such that their product is again the world-set relation. A possible world can be constructed by choosing one tuple from each relation. The decomposition is based on interdependence between sets of fields within a tuple; dependent fields are put in the same component, independent fields are put in different components.

2.9 Fuzzy data

Fuzzy set and possibility theory are closely related terms. A fuzzy set is an extension of a standard set. With fuzzy sets, each element is associated with a value between 0 and 1 indicating the weight of the membership of the element in the fuzzy set. The central notion in possibility theory is that of the possibility distribution. The possibility distribution may be used to specify to which extent elements of a particular set are allowed as the actual value of a variable x. An example of a data model, that is based on fuzzy set theory, can be found in [7]. It supports the modeling of gradual properties, vague classes and approximate descriptions. The data model does have support for representing constructs like ”a person A aged 40 belongs to the class ”middle-aged” people with a degree of .8” in relations. So, an item is assigned to a set with the help of membership function, the assignment is stored in the database along with a probability indicating the confidence of the assignment. A derivation of fuzzy set theory is fuzzy logic which deals with reasoning that is approximate instead of precise (predicate logic). Household appliances, air conditioners and vehicle subsystems are examples in which fuzzy logic is used. A washing machine, for instance, senses the amount of laundry and adjusts the amount of water and detergent to be used accordingly. The microchip in the washing machine has to make decisions based on the variable ”amount of laundry”. The variable ”amount of laundry” can be subdivided into states: ”not much laundry”, ”some laundry”, ”normal amount”, ”a lot of laundry”. The transition from one state to another is hard to define. The term fuzzy database is commonly used when people refer to information systems that store data in fuzzy set-based relations. Actually, this term has several meanings. We refer to [7, 6] for a more comprehensive overview.

Chapter 3

Modeling Continuous Uncertain Data

In this chapter the data model for continuous uncertain data will be discussed. The data model supporting continuous uncertainty is based on the Probabilistic XML data model proposed in [12, 23]. First, the properties of this Probabilistic XML data model are presented. Next, another topic which will be discussed is the interpretation of a continuous distribution in terms of possible worlds. One of the last sections of this chapter is dedicated to the specification of the model to be used for representing continuous uncertain data. This chapter contains several definitions. These definitions are either directly taken from [12, 23], or adapted, based on the definitions presented in [12, 23].

3.1 Probabilistic XML

3.1.1 Possible worlds

The probabilistic XML data model proposed in [12, 23] is based on the possible world semantics explained in 2.2. The possible world representation of the probabilistic XML data model enumerates all possible worlds and encode them in an XML document as separate subtrees. Each subtree is associated with a probability.

Table 3.1 shows a simplified version of a database used in a patient record system. The table shows the relation between name and birthdate. There is uncertainty in this relation. This uncertainty is represented in possible worlds in table 3.2. Figure 3.1 shows the encoding of the enumeration of possible worlds in a tree.

When a database stores data using the possible world representation, a lot of redundant data will be stored in the database. As can be seen in table 3.2, there is overlap in names

Table 3.1: Part of the data in a Patient Record System

Name	**Birthdate**
Jan Janssen	8-3-1983
Jan Janssen	9-3-1983
Piet Bakker	4-1-1976
Piet Bakker	4-2-1976

Table 3.2: Construction of Possible Worlds

World 1

Name	**Birthdate**
Jan Janssen	8-3-1983
Piet Bakker	4-1-1976

World 2

Name	**Birthdate**
Jan Janssen	8-3-1983
Piet Bakker	4-2-1976

World 3

Name	**Birthdate**
Jan Janssen	9-3-1983
Piet Bakker	4-1-1976

World 4

Name	**Birthdate**
Jan Janssen	9-3-1983
Piet Bakker	4-2-1976

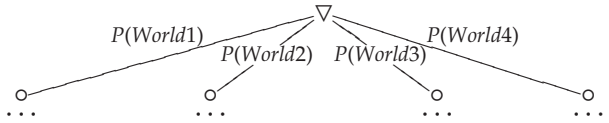


Figure 3.1: *Possible world representation of Patient Record Example (XML), figure has been taken from [12].*

and birthdates. Each unique pair of names and birthdates is duplicated across some of the worlds. Although, thinking and reasoning in terms of possible worlds is sound, enumerating possible worlds and encoding them in an XML document is not practical. For that reason, the Compact representation was proposed in [12, 23]. This type of representation will be discussed in the next section.

3.1.2 Compact representation

The work presented in this section is based on the possible world semantics discussed earlier. The compact representation encodes discrete uncertainty in an XML document and reduces the storage of redundant data compared to the approach discussed in the previous section. In order to reduce the storage of redundant data, the compact representation distinguishes three different kinds of nodes: possibility nodes, probability nodes and regular XML nodes. Children of probability nodes are possibility nodes which represent different possibilities or alternatives, possibility nodes encodes encode probabilities. The children of possibility nodes are XML nodes. The regular XML nodes contain the base data. This approach leads to a reduction of overlapping data as we will see later.

In contrast to set theory and relational data, order is important in XML documents. In the formal definitions which will be presented in the next few section, set-based theory is used. For this reason, some notations for handling sequences will be introduced first.

Notational convention 1 *Analogous to the powerset notation $\mathbb{P}$, we use a* power sequence *notation $\mathbb{S}\,A$ to denote the domain of all possible sequences built up of elements of A. We use the notation $[a_1, \ldots, a_n]$ for a sequence of n elements $a_i \in A$ $(i = 1..n)$. We use set operations for sequences, such as $\cup, \exists, \in$, whenever definitions remain unambiguous.*

The notions of a tree and subtree are abstractions from the XML document and XML fragment, respectively. A tree is modeled as a node having a sequence of subtrees as children of the node.

Definition 2 *Let $n = \{id, tag, kind, attr, value\}$ be a node, with*

- *id the node identity*
- *tag the tag name of the name*
- *kind the node kind*
- *attr the list of attributes, which can be empty*
- *value the text value of the node, which can be empty*

In an implementation, a node is defined in terms of their properties like in definition 2. In this theoretical model, we abstract from this definition. In the following definition, the notion of a tree is defined.

Definition 3 *Let $\mathcal{N}$ be the set of* nodes. *Let $\mathcal{T}_i$ be the set of* trees *with maximum level i inductively defined as follows:*

$$\begin{aligned}
\mathcal{T}_0 &= \{(n, \emptyset) \mid n \in \mathcal{N}\} \\
\mathcal{T}_{i+1} &= \mathcal{T}_i \cup \{(n, ST) \mid n \in \mathcal{N} \\
&\qquad \wedge ST \in \mathbb{S}\,\mathcal{T}_i \\
&\qquad \wedge (\forall T \in ST \bullet n \notin \mathcal{N}^T) \\
&\qquad \wedge (\forall T, T' \in ST \bullet T \neq T' \\
&\qquad\quad \Rightarrow \mathcal{N}^T \cap \mathcal{N}^{T'} = \emptyset)\}
\end{aligned}$$

where $\mathcal{N}^T = \{n\} \cup \bigcup_{T' \in ST} \mathcal{N}^{T'}$. Let $\mathcal{T}_{\text{fin}}$ be the set of finite trees, *i.e., $T \in \mathcal{T}_{\text{fin}} \Leftrightarrow \exists i \in \mathbb{N} \bullet T \in \mathcal{T}_i$. In the sequel, we only work with finite trees.*

Operations are frequently performed on children of a node or a subtree instead of an entire tree or a single node. For this reason, functions to obtain subtrees or children are defined. A subtree can be retrieved by performing the function subtree with parameters T, indicating the tree containing the subtree, and n, indicating the rootnode of the obtained subtree. The function child accepts the parameters T, indicating the tree containing the subtree, and n, indicating the parent of the returned nodes.

Definition 4 *Let* $\mathsf{subtree}(T, n)$ *be the* subtree *within $T = (\overline{n}, ST)$ rooted at n.*

- $\mathsf{subtree}(T, n) = \begin{cases} T & \text{if } \overline{n} = n \\ \mathsf{subtree}(T', n) & \text{otherwise} \end{cases}$

where T' such that $(n', T') \in ST \wedge n \in \mathcal{N}^{T'}$.
For $\mathsf{subtree}(T, n) = (n, [(n_1, ST_1), \ldots, (n_m, ST_m)])$,
let $\mathsf{child}(T, n) = [n_1, \ldots, n_m]$.

The central notion in the compact representation is that of the *probabilistic tree*. In an ordinary XML document all information is certain, whereas in a probabilistic XML document each XML node has one or more possibilities or alternatives. As mentioned before, the compact representation introduces two new kinds of nodes:

1. probability nodes depicted as $\triangledown$, and

2. possibility nodes depicted as ◦, which have an associated probability.

In probabilistic XML, ordinary XML are depicted as •. The root of a probabilistic XML document is always a probability node. The children of a probability nodes are always possibility nodes, children of possibility nodes are always ordinary XML nodes, and children of these XML nodes are probability nodes. The result of using this layered structure is a well-structured probabilistic tree where each level of the tree only contains one kind of nodes.

For examples of probabilistic XML documents and applications of probabilistic XML, we refer to [12]. [12] contains a formal definition of a probabilistic tree, a formal definition for deriving possible worlds from a probabilistic tree and it contains an algorithm for counting the number of possible worlds encoded in a probabilistic tree. We chose not to include these definitions here as we will present adapted versions of these definitions in one of the following sections. The adapted definitions allow the encoding of continuous uncertainty in a probabilistic tree using possible world semantics. However, before we can elaborate on the representation of continuous uncertainty in a probabilistic tree, we have to agree on how continuous uncertainty can be expressed in terms of possible worlds.

3.2 Expressing Continuous Uncertainty in Possible Worlds

The Gaussian, gamma and continuous uniform distributions are all examples of absolutely continuous distributions. A probability distribution is called absolutely continuous if it has an associated probability density function. An equivalent definition of an absolutely continuous distribution is that the probability equals zero if a random variable X attains a value on any given interval.

Figure 3.2 shows the probability density function of a stochastic variable that has a Gaussian distribution. The stochastic variable is, for instance, the room temperature; the distribution has parameters mean 18 and variance 2. A property of the probability density function is that the integral of a probability density function equals 1. Another property is that for all values of $x, f(x)$ is greater than or equal to zero.

The probability density function that is shown in figure 3.2 can be sampled. Sampling the probability density function on the interval $[13, 23]$, yields a histogram which approximates the continuous distribution by means of bars. The Middle Riemann Sum method is used here for the approximation of the probability density function. The width of each bar corresponds to the *unit* or the *segment* size. In this example each unit has a size of 2. Figure 3.3 depicts this sampling. The areas of the bars correspond to approximations of the following probabilities: $P(13 \leq X < 15)$, $P(15 \leq X < 17)$, $P(17 \leq X < 19)$, $P(19 \leq X < 21)$, $P(21 \leq X < 23)$. When the probability density function is sampled on the same interval with a smaller segment size, the resolution of the histogram increases. This means that the histogram consists of a larger number of segments, each segment having a smaller interval. Each bar has a smaller area and thus denotes a smaller probability. In a very fine-grained histogram, the segment size becomes close to zero, the area of each bar (column) approaches zero meaning that the probability approaches zero. Moreover, the number of segments approaches infinity. Thus, the result of sampling is histogram which approaches the curve of a probability density function.

In contrast to figure 3.3, probability density functions in general and especially the Gaussian distribution, are not bounded on an interval. The continuous uniform distribution is an example of a distribution that is defined on a specific interval. Thus, theoretically, the result

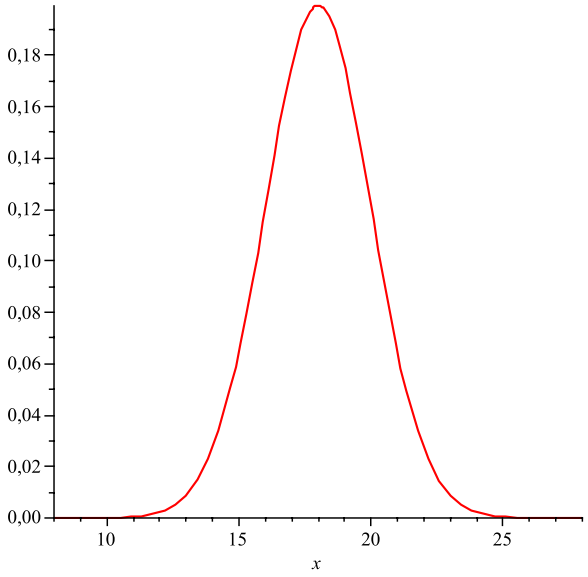


Figure 3.2: *Probability density function of a Gaussian distribution with parameters mean 18 and variance 2.*

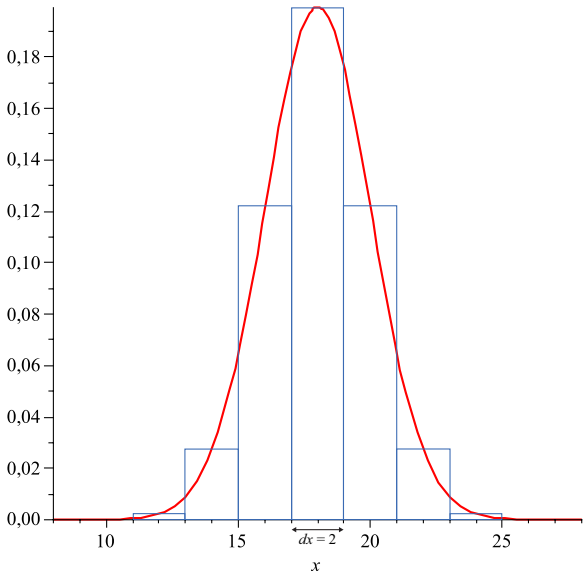


Figure 3.3: *An approximation of the area under the probability density function by means of rectangles.*

of sampling a continuous distribution is, in most cases, a very fine-grained histogram which is unbounded.

When a database models a particular aspect of the real world, the database holds information on real world objects. In case of an uncertain database, a database holds information on possible representations of real world objects. As mentioned before, a possible representation is also called a possibility. A real world object that can be described by a continuous distribution, is represented in a continuous uncertain database as an infinite sequence of possibilities. Each element in the set of possibilities has a value and an associated probability approaching zero. This all follows from the sampling approach explained before, if we consider a possibility to be equivalent to a segment and the associated probability to be equivalent to the segment area under the graph of a probability density function. Consider again figure 3.3. The probability density function is sampled, yielding seven histogram segments or seven possibilities. Each possibility has an interval, a value (the middle of the interval) and a probability (the area of the column). The equivalence between a continuous distribution and an enumeration of possibilities can be described by the following equation:

The representation in a probabilistic database of a real world object, o, that can be described by a continuous distribution, is defined as follows:

- $\mathsf{distribution}(o) \leftrightarrow (S = [s_1, \ldots, s_\infty])$
- $\forall s_i \in S \wedge \exists r \in \mathbb{R} \Rightarrow \mathsf{prob}(s_i) \approx 0 \wedge \mathsf{value}(s_i) = r$

where distribution *samples the continuous distribution into possibilities, s_i, S denotes the set of possibilities,* prob *returns the probability which is associated with each possibility and* value *returns the value that corresponds with a possibility.*

As mentioned before, a possible world is constructed by choosing one alternative out of a set of alternatives for each of the real world objects represented by the database (and the non-existence of an rwo is to be considered an alternative as well). When a database contains the representation of one or more real world object(s) that can be described by continuous distributions, the database holds an infinite number of possibilities. In that situation, the database also holds an infinite number of possible worlds.

As was shown in section 3.1, uncertainty can be represented in XML by enumerating possible worlds and encode them in separate subtrees. Following this approach when continuous uncertain data is involved, yields a tree as shown in figure 3.4. This approach is theoretically very interesting, though practically not feasible. Besides the problem of duplicate information across possible worlds as discussed in the previous section, the enumeration and encoding of an infinite number of possible worlds into separate subtrees would yield a never-ending XML document. For this reason, the continuous representation will be introduced.

3.3 Representing Continuous Uncertainty

In this section the continuous representation, used to model continuous uncertain data, will be presented. The continuous representation solves some limitations of the compact representation. The representation introduced here is entirely based on the compact representation proposed in [12, 23].

The central notion in the compact as well as in the continuous representation is the

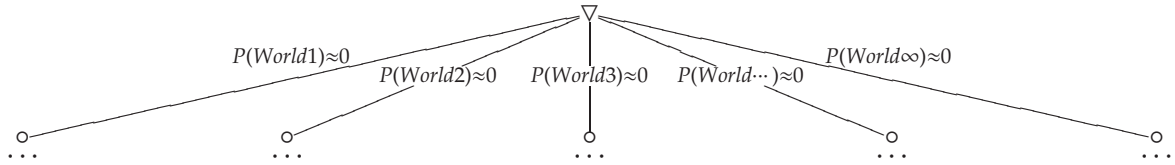


Figure 3.4: *Possible world representation of continuous uncertain data.*

probabilistic tree. As discussed before, a probabilistic XML document consists of probability nodes, possibility nodes and regular XML nodes. Possibility nodes are used to model zero or more alternatives for an XML node. In the compact representation, which can be used to model discrete uncertainty, each subtree of a possibility node denotes one possibility.

The continuous representation is somewhat more complicated with respect to this. When the continuous representation is used to model continuous uncertain data, possibility nodes are associated with continuous distributions and with the probability mass, in other words: possibility nodes hold probability density functions and the probability mass. Consequently, each possibility node represents an infinite number of possibilities. The continuous representation handles discrete probabilities in the same way as the compact representation does. In the compact representation, each possibility node has an associated probability. The continuous representation also uses possibility nodes with associated probabilities when modeling discrete uncertainty. The continuous as well as the compact representation require that the total probability mass under a probability node sums up to 1. This means that the sum of the the probabilities associated with possibility nodes and the area below the graph of a probability density function have to be equal to 1.

In the continuous representation, the possibility nodes may encode probability density functions along with its parameters. We need to define which probability density functions are supported by the probabilistic model.

Definition 5 *The supported probability density functions are defined by the following set:*

- $\mathsf{PDF}_0 = \{\mathsf{gaussian}(\mu, \sigma^2),\ \mathsf{gamma}(k, \theta),\ \mathsf{uniform}(a, b),\ \mathsf{beta}(\alpha, \beta)\}$
- $\mathsf{PDF}_i = \mathsf{PDF}_{i-1} \cup \bigcup_{p \in \mathsf{PDF}_{i-1}} \mathsf{floor}(p, F)$
- $\mathsf{PDF} = \bigcup_i \mathsf{PDF}_i$

In definition 5, PDF_0 is the set of labels that refer to the probability density functions of standard continuous distributions. Besides the probability density functions, the labels also specify the parameters of the probability density function, e.g. μ and σ for the Gaussian distribution. The set PDF_i contains the labels that refer to the probability density functions of non-standard continuous distributions. An example of a non-standard continuous distribution is a continuous distribution that has been floored. The application of a floor over a standard continuous distributions acts as a selection predicate or a filter. In terms of semantics, the possibilities that do not satisfy the selection criteria are filtered out. Consequently, these possibilities do not exist in the resulting continuous distribution. The area under the curve of the resulting probability density function is less than one. Thus, in the result, probability mass is missing. A floor is used to cut-off a continuous distribution. An example of a situation in which this operator is useful, is the representation of sensor readings. In many cases, sensor readings are only correct if the reading falls within a particular interval which

is part of the whole domain. The floor operator can be used to create a new probability distribution function that is specified on that particular interval.

A probabilistic tree is defined as a tree, a kind function that assigns node kinds to specific nodes in the tree, a prob function which attaches probabilities to possibility nodes and a distribution function which attaches probability density functions to possibility nodes. The root node is defined to always be a probability node. A special type of probabilistic tree is a *certain* one, which means that all information in it is certain, i.e., all probability nodes have exactly one possibility node with an associated probability of 1.

Definition 6 *A* probabilistic tree $PT = (T, \mathsf{kind}, \mathsf{prob}, \mathsf{distribution})$ *is defined as follows*

- $\mathsf{kind} \in (\mathcal{N} \rightarrow \{\underline{\mathsf{prob}}, \underline{\mathsf{poss}}, \underline{\mathsf{xml}}\})$
- $\mathcal{N}_k^T = \{n \in \mathcal{N}^T \mid \mathsf{kind}(n) = k\}$.
- $\mathsf{kind}(\overline{n}) = \underline{\mathsf{prob}}$ *where* $T = (\overline{n}, ST)$
- $\forall n \in \mathcal{N}_{\underline{\mathsf{prob}}}^T \forall n' \in \mathsf{child}(T, n) \bullet n' \in \mathcal{N}_{\underline{\mathsf{poss}}}^T$
- $\forall n \in \mathcal{N}_{\underline{\mathsf{poss}}}^T \forall n' \in \mathsf{child}(T, n) \bullet n' \in \mathcal{N}_{\underline{\mathsf{xml}}}^T$
- $\forall n \in \mathcal{N}_{\underline{\mathsf{xml}}}^T \forall n' \in \mathsf{child}(T, n) \bullet n' \in \mathcal{N}_{\underline{\mathsf{prob}}}^T$
- $\mathsf{prob} \in \mathcal{N}_{\underline{\mathsf{poss}}}^T \rightarrowtail [0, 1]$
- $\mathsf{distribution} \in \mathcal{N}_{\underline{\mathsf{poss}}}^T \rightarrowtail \{\mathsf{discrete}\} \cup \mathsf{PDF}$
- $\forall s \in \mathrm{dom}(\mathsf{distribution}(s)) \bullet \mathsf{distribution}(s) \in \mathsf{PDF} \implies s \in \mathrm{dom}(\mathsf{prob}) \wedge \mathsf{prob}(s) = \int_{-\infty}^{+\infty} \mathsf{pdf}(s)(x)dx$
- $\forall n \in \mathcal{N}_{\underline{\mathsf{prob}}}^T \bullet ((\sum_{n' \in \mathsf{child}(T,n)} \mathsf{prob}(n')) = 1 \vee (\forall n' \in \mathsf{child}(T, n) \bullet \mathsf{prob}(n') = \bot))$.

Where $A \rightarrowtail B$ *is a partial function.*

A probabilistic tree $PT = (T, \mathsf{kind}, \mathsf{prob}, \mathsf{distribution})$ *is* certain *iff there is only one possibility node for each probability node and the possibility node is not associated with a continuous distribution, i.e.,* $\mathsf{certain}(PT) \Leftrightarrow \forall n \in \mathcal{N}_{\underline{\mathsf{prob}}}^T \bullet |\mathsf{child}(T, n)| = 1$.*To clarify definitions, we use b to denote a probability node, s to denote a possibility node, and x to denote an XML node.*

Continuous probability distributions can be characterized by probability density functions. A probability density function is defined as a function of a possibility node holding the kind of probability density function with its parameters, and the input variable x which is an element from the set $\mathbb{R}$. The result is also a rational number. The attentive reader will notice that definition 7 allows the modeling of continuous probability distributions that have been floored.

Definition 7 *Let* $\mathsf{pdf}(s)(x)$ *return the value of the probability density function, which is associated with possibility node* s*, with as input* x*. Since function* pdf *is a probability density function describing the probability density in terms of the input variable* x*, it has the following properties:*

- $\mathsf{pdf} \in \mathcal{N}_{\underline{\mathsf{poss}}} \rightarrowtail \mathbb{R} \rightarrow \mathbb{R}$
- $\forall s \in \mathrm{dom}(\mathsf{pdf}) \forall x \in \mathbb{R} \bullet \mathsf{pdf}(s)(x) \geq 0$
- $\forall s \in \mathrm{dom}(\mathsf{distribution}(s)) \bullet$

$$\mathsf{pdf}(s) = \begin{cases} \text{"usual probability density function} \\ \text{associated with } \mathsf{distribution}(s)\text{"}, & \text{if } \mathsf{distribution}(s) \in \mathsf{PDF}_0 \\ \lambda x : \mathbb{R} \bullet \begin{cases} 0, & \text{if } x \in F \\ f(x), & \text{otherwise} \end{cases} & \text{if } \mathsf{distribution}(s) = \mathsf{floor}(f, F) \\ \bot & \text{otherwise} \end{cases}$$

A probability density function can be used to compute probabilities. A real-world object can be described by a random variable whose probability distribution is continuous. The probability that a random variable attains a value less or equal than a given value, x, is defined by the following definition.

Definition 8 *The real world object,* o*, can be described by the continuous random variable* X*. Let* $\mathsf{Pr}(s, X \leq x)$ *be the probability that the continuous random variable attains* X *a value less or equal* x*.*

- $\mathsf{Pr}(s, X \leq x) = \int_{-\infty}^{x} \mathsf{pdf}(s)(x)dx$

where s *is a possibility node, encoding the probability density function of the continuous random variable* X*.*

A possibility node that encodes a probability density function represents an infinite number of possibilities, the possibility node can be replaced by an infinite sequence of possibility nodes. By doing this, we *expand* or *slice* the possibility node that has an associated probability density function. The parent node of the resulting sequence is the probability node which can be seen as the root node of a probabilistic (sub-)tree. The children of the probability node is a sequence of possibility nodes which may contain possibility nodes that represent probabilities (discrete uncertain data) as well as an infinite number of possibility nodes which correspond to one or more continuous distributions. The possibility nodes of the latter type, have an associated probability approaching zero. This is more formally described by the following definition.

Definition 9 *Let* $\hat{\alpha}_{s'}$ *be the probability associated with possibility node* s' *after expanding the probability density function encoded by possibility node* s*.*

- $\hat{\alpha}_{s'} = \mathsf{Pr}(s, X \leq x + dx) - \mathsf{Pr}(s, X \leq x)$

After expanding the possibility node, each resulting possibility node has one child which is a regular XML node holding a value. A possibility node holding a continuous distribution can be transformed from the continuous representation to the compact representation using the expand function.

Definition 10 *Let* $\mathsf{expand}(T, n)$ *be a probabilistic subtree of* T *rooted at* n *enumerating the infinite number of possibilities summarized by a possibility node that encodes a probability density function and is originally a child of* n*. Each resulting possibility is associated with a probability approaching zero, i.e.,* $\mathsf{prob}(s'_i) = \hat{\alpha}_{s'_i}$*.*

$$\mathsf{expand}(T, n) = \begin{cases} (n, D \cup [(s'_1, [x'_1]), \ldots, (s'_q, [x'_q])]), & \text{if } \exists s \in \mathsf{child}(T, n) \bullet \\ & \mathsf{distribution}(s) \in \mathsf{PDF} \\ \mathsf{subtree}(T, n), & \text{otherwise} \end{cases} \tag{3.1}$$

where $n \in \mathcal{N}^T_{\underline{\mathsf{prob}}} \wedge s' \in \mathcal{N}_{\underline{\mathsf{poss}}} \wedge x' \in \mathcal{N}_{\underline{\mathsf{xml}}} \wedge \mathsf{prob}(s'_i) = \hat{\alpha}_{s'_i} \wedge D = [\forall c \in \mathsf{child}(T, n) | \mathsf{distribution}(c) \notin \mathsf{PDF} \bullet c] \wedge \{x'_1, \ldots, x'_q\} = \mathrm{dom}(\mathsf{pdf}(s)) \wedge \forall i, j \in \mathbb{N} \bullet i \neq j \implies s_i \neq s_j$

A probability density function is defined on an interval. This interval is open or it is bounded. In both situations, when dx approaches zero, the associated $\hat{\alpha}_{s'}$ approaches zero and the number of possibility nodes approaches infinity. Consequently, the number of XML nodes, that are children of the possibility nodes, approaches infinity. This is similar to the sampling approach which we described in section 3.2. When the size of a segment in a histogram approaches zero, the area of that segment approaches zero. In order to fill the curve of a probability density function with segments, an infinite number of segments is required.

A probabilistic tree may contain several possibility nodes, each encoding a probability density function. Applying the expand on the parents, the probability nodes, of those possibility nodes yields a new probabilistic tree. The resulting probabilistic tree is the compact representation of the set of possible worlds. In this probabilistic tree, each probability node representing continuous uncertainty, have an infinite number of possibility nodes as children. In the compact representation, a possible world can be constructed from a probabilistic tree by making a decision at each probability node for one of the possibility nodes. For this reason, a probability node can also be seen as a *choice point*. Each possible world can be represented by a certain probabilistic tree. Definition 11 assumes that all possibility nodes in PT have already been expanded.

Definition 11 *A certain probabilistic tree* PT' *is a* possible world *of another probabilistic tree* PT*, i.e.,* $\mathsf{pw}(PT', PT)$*, with* probability $\mathsf{pwprob}(PT', PT)$ *iff*

- $PT = (T, \mathsf{kind}, \mathsf{prob}) \wedge PT' = (T', \mathsf{kind}', \mathsf{prob}')$
- $T = (\overline{n}, ST_{\overline{n}}) \wedge T' = (\overline{n}, ST'_{\overline{n}})$
- $\exists s \in \mathsf{child}(T, \overline{n}) \bullet \mathsf{child}(T', \overline{n}) = [s]$
- $X = \mathsf{child}(T, s) = \mathsf{child}(T', s)$
- $\forall x \in X \bullet \mathsf{child}(T, x) = \mathsf{child}(T', x)$
- $B = \bigcup_{x \in X} \mathsf{child}(T, x)$
- $\forall b \in B \bullet PT_b = \mathsf{subtree}(PT, b)$
  $\wedge PT'_b = \mathsf{subtree}(PT', b)$
  $\wedge \mathsf{pw}(PT'_b, PT_b)$
- $\forall b \in B \bullet p_b = \mathsf{pwprob}(PT_b, PT'_b)$
- $\mathsf{pwprob}(PT', PT) = \mathsf{prob}(s) \times \prod_{b \in B} p_b$

The set of all possible worlds of a probabilistic tree PT *is*
$\mathsf{PWS}_{PT} = \{PT' \mid \mathsf{pw}(PT', PT)\}$.

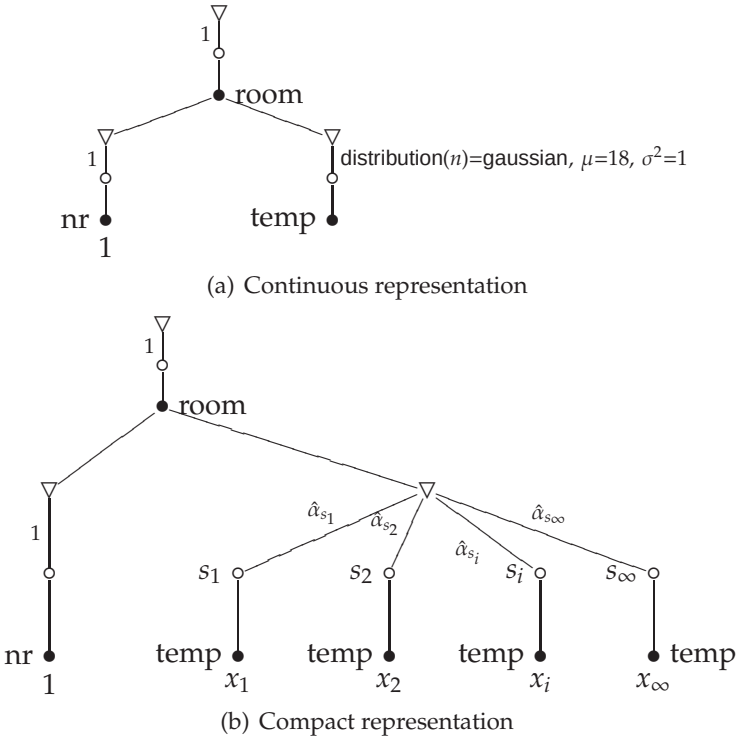


Figure 3.5: *Examples of the continuous and the compact representations.*

The number of possible worlds captured by a probabilistic tree is determined by the number of probability nodes and the number of possibilities below each probability node. We also define a function leaf that returns all the leaf nodes of a tree.

Definition 12 *In the continuous representation, the number of possible worlds defined by the tree PT, $N_{PT}^{PW(T)}$ is equal to the number of possible worlds at node $\overline{n}$, defined by $N_{\overline{n}}^{PW(T)}$ where*

- $\mathsf{leaf}(T) = \{n | n \in \mathcal{N}^T \bullet \mathsf{child}(n) = \emptyset\}$
- $N_n^{PW(T)} = 1$, *if* $n \in \mathsf{leaf}(T)$
- $N_n^{PW(T)} = \prod_{n' \in \mathsf{child}(T,n)} N_{n'}^{PW(T)}$, *if* $\mathsf{kind}(n) = \underline{\mathsf{poss}} \wedge \mathsf{distribution}(n) \notin \mathsf{PDF}$
- $N_n^{PW(T)} = \infty$, *if* $\mathsf{kind}(n) = \underline{\mathsf{poss}} \wedge \mathsf{distribution}(n) \in \mathsf{PDF}$
- $N_n^{PW(T)} = \sum_{n' \in \mathsf{child}(T,n)} N_{n'}^{PW(T)}$, *if* $\mathsf{kind}(n) = \underline{\mathsf{prob}}$
- $N_n^{PW(T)} = \prod_{n' \in \mathsf{child}(T,n)} N_{n'}^{PW(T)}$, *if* $\mathsf{kind}(n) = \underline{\mathsf{xml}}$

Using the definition above, one can calculate the number of possible worlds captured by a probabilistic tree. The definition says that if a tree captures at least one continuous distribution, the number possibilities and thus also the number of possible worlds it represents is ∞.

3.4 Examples

Figure 3.5 shows an example of how a continuous distribution can represented in a probabilistic database. In the example, the temperature in a room of an office building during the day can be described by a continuous distribution. The database stores the combinations of room numbers and the corresponding temperature models for the whole office building. The continuous representation shown in figure 3.5(a) stores the probability density function with its parameters. Figure 3.5(b) shows the corresponding expanded version of the probabilistic tree from figure 3.5(a), this representation is also called the compact representation.

3.5 Conclusion

In this chapter, we introduced a probabilistic data model that supports the representation of continuous uncertain data in XML. We explained how continuous uncertain data can be expressed in possible worlds. Besides that, we proposed the continuous representation which is able to represent continuous uncertain data in XML in an efficient way. The continuous representation is an extension to the compact representation proposed in [12, 23]. Furthermore, we showed that the continuous representation and the compact representation are equivalent since the continuous representation can be transformed to the compact representation using the expand(T, n) function. As a result of this, all the properties that are valid for the compact representation, are also valid for the continuous representation.

Chapter 4

Query language

One of our goals is to provide simple, yet very powerful, constructs for querying XML documents containing continuous uncertain data. In conjunction with the XQuery language, the constructs can be used by the end-user to compose queries. In this chapter we will discuss querying techniques in terms of possible world semantics. First, we need to show how the semantics are followed when querying continuous uncertain data. Then, we will explain the working of basic operators, such as querying the probability or retrieving the mean from a distribution. We conclude this chapter with a discussion on advanced functions that are capable of aggregating one or more continuous distributions in different ways.

4.1 Possible worlds

In the previous chapter we examined two representations in detail: the compact and the continuous representation. As we discussed in chapter 2, most uncertain database management systems are based on the possible world semantics.

Pirotte and Zimányi showed in [27] how the semantics are followed when querying an uncertain database. Information residing in a database is represented by a set of possible worlds. Following the possible world semantics, querying uncertain data means that a query is evaluated in every possible world captured in a probabilistic database, the results of the individual query evaluations are collected and the union of these results are presented to the user as if it is one result.

Section 3.1.1 showed that a collection of possible worlds stored in a probabilistic database, contains overlapping data. Moreover, in terms of semantics, continuous uncertain data is represented by an infinite number of possible worlds. Querying redundant or overlapping data is not efficient and querying continuous uncertain data by enumerating an infinite number of possible worlds is infeasible. For these two reasons, the query operations we will propose will work directly on the continuous representation. The continuous representation should be fully compatible with possible world semantics because it only extends the compact representation in such a way that it supports the modeling of continuous uncertainty; [12, 23] showed that the compact representation is consistent with possible world semantics when querying discrete uncertain data.

Figure 4.1 shows a commutative diagram. D is a single database, $D_1 ... D_\infty$denote possible worlds and R is a function which returns the query result. The top part of the figure represents the continuous representation and the lower part represents the possible world semantics.

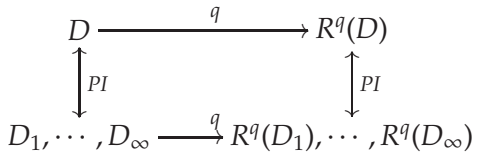


Figure 4.1: *Commutative diagram, figure has been taken from [12].*

The figure also shows two vertical arrows. These arrows denote the transformation between the possible worlds and the continuous representation (and vice versa).

4.2 Querying a continuous distribution

An XML document containing continuous uncertain data holds multiple continuous distributions. Each possible world stored in the document, consists of one possibility (a slice) from each continuous distribution encoded in the document. Following the possible world semantics, computing a probability is an operation that works across possible worlds because a probability is associated with a possible world and is not known within a possible world. Examples of other operations that are useful and work across possible worlds, include the computation of the mean and the variance of a continuous distribution. All these value-based query operators do not conform to possible world semantics. In this section we will define several probabilistic query operations using probability theory as basis. As discussed in the previous section, we will define these operations in such way that they can be used on the continuous representation.

4.2.1 The mean function

The expected value is a way to describe the location of the distribution in the set of possible values (which is infinite in the case of a continuous distribution) in terms of the location of the probability mass. It is computed by taking the integral over the input values of the probability density function multiplied with the outcome of the probability density function for each element in the set of possible values. A more formal description is given in definition 13.

Definition 13 *Let* $\mathsf{mean}(T, s)$ *be a function that computes the expected value of the continuous distribution encoded in possibility node, s.*

- $\mathsf{mean}(T, s) = \int_{-\infty}^{+\infty} x * \mathsf{pdf}(s, x)dx$

where $x \in \mathbb{R}$ *and T is the probabilistic tree containing the possibility node s.*

4.2.2 The variance function

In statistics, the variance of a distribution is a way to express the degree of being spread out. The variance is computed by averaging the squared distances of the possible values from the expected value. A more formal description is given below.

Definition 14 *Let* $\mathsf{variance}(T, s)$ *be a function that computes the variance of the continuous distribution encoded in possibility node, s.*

- $\mathsf{variance}(T, s) = \int_{-\infty}^{+\infty} \mathsf{pdf}(s, x) * (x - \mathsf{mean}(T, s))^2 dx$

where $x \in \mathbb{R}$ *and T is the probabilistic tree containing the possibility node, s.*

4.2.3 The vmin function

The vmin function computes the minimum possible value of a continuous distribution. A more formal description is given below.

Definition 15 *Let* $\mathsf{vmin}(T, s)$ *be a function that computes the minimum value of the continuous distribution encoded in the possibility node, s.*

$$\mathsf{vmin}(T, s) = \begin{cases} -\infty & \text{if } \mathsf{distribution}(s) = \mathsf{gaussian} \\ 0 & \text{if } \mathsf{distribution}(s) = \mathsf{gamma} \bigvee \mathsf{distribution}(s) = \mathsf{beta} \\ a & \text{if } \mathsf{distribution}(s) = \mathsf{uniform} \end{cases} \tag{4.1}$$

where T is the probabilistic tree containing the probability node s.

4.2.4 The vmax function

The vmax function computes the maximum possible value of a continuous distribution. A more formal description is given below.

Definition 16 *Let* $\mathsf{vmax}(T, s)$ *be a function that computes the maximum value of the continuous distribution encoded in the possibility node, s.*

$$\mathsf{vmax}(T, s) = \begin{cases} +\infty & \text{if } \mathsf{distribution}(s) = \mathsf{gaussian} \bigvee \mathsf{distribution}(s) = \mathsf{gamma} \\ 1 & \text{if } \mathsf{distribution}(s) = \mathsf{beta} \\ b & \text{if } \mathsf{distribution}(s) = \mathsf{uniform} \end{cases} \tag{4.2}$$

where T is the probabilistic tree containing the possibility node s.

4.2.5 Predicates

In the previous sections we discussed the behavior of several operators that process one continuous distribution returning one single value as a result. These operators can be used as means to determine the properties of one or more distributions by simply executing the operator over each distribution and enumerating the results. Since these operators are able to determine the properties of distributions, they can also be used in a predicate. A predicate can be composed using one or more operators and all distributions satisfying the predicate will be returned. Consider figure 4.2, it shows a part of the sensor database which is in use by the Dutch Environment Agency. It is an example of a semi-structured

```
<locations>
    <location>
        <name>Arnhem</name>
        <prob>
          <poss type=gaussian(920,20)>
            <water-level/>
          </poss>
        </prob>
    </location>
    <location>
        <name>Den Helder</name>
        <prob>
          <poss type=gaussian(-30,5)>
            <water-level/>
          </poss>
        </prob>
    </location>
</locations>
```

Figure 4.2: *An example of a sensor database.*

Table 4.1: The value-probability pairs correspond to the segments in histograms A and B.

(a) Histogram H_A

Segment	Value	Probability
g_{A1}	2	0.1079819330
g_{A2}	4	0.7978845605
g_{A3}	6	0.1079819330

(b) Histogram H_B

Segment	Value	Probability
g_{B1}	10	0.1079819330
g_{B2}	12	0.7978845605
g_{B3}	14	0.1079819330

database containing continuous uncertain data. The data is originating from sensors that measure the actual water levels on different locations in The Netherlands. The Dutch Environment Agency is responsible for notifying several governmental organizations when the water level reaches a certain limit. For this reason, an employee of the Dutch Environment Agency wants to know the locations where the expected value of the water level is more than 100 centimeters. In order to get this knowledge, he poses the following query to the database: `/location[mean(water-level)>100]/name`. The answer to this query will be `<name>Arnhem</name>`.

4.3 Aggregating continuous distributions

Aggregation operators can be used to aggregate multiple continuous distributions in one single continuous distribution. If a distribution is considered to describe a real world object, those descriptions use the same unit of measurement, and the distributions are *independent*, an aggregation function can be used to aggregate descriptions of similar real world objects. The resulting distribution can be presented to the end-user or used for further processing. We will show that these aggregation query operators follow the possible world semantics

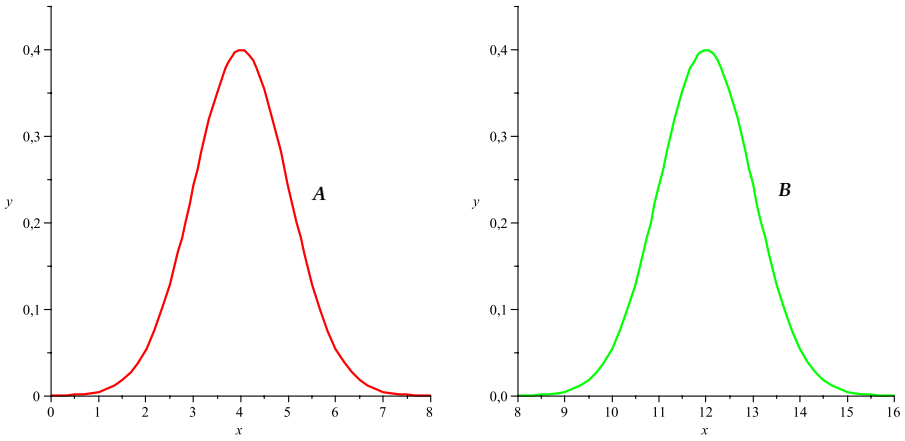
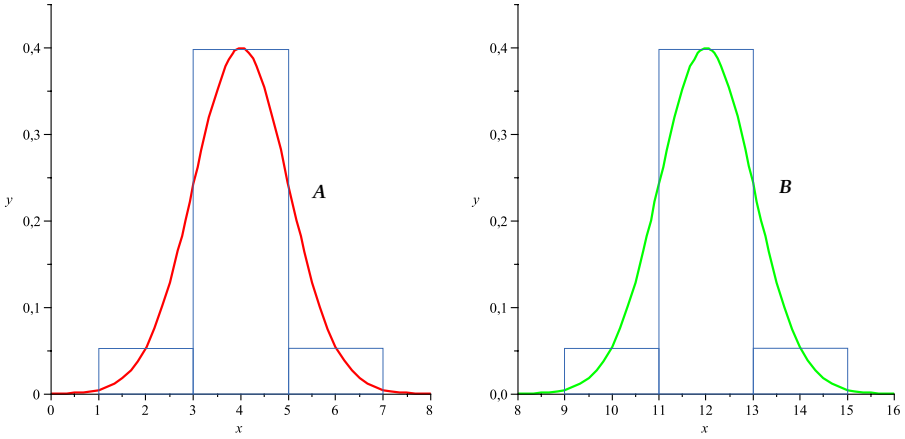


Figure 4.3: *The probability density functions of Gaussian distributions A and B.*



(a) Histogram A (b) Histogram B

Figure 4.4: *The sampled probability density functions.*

Table 4.2: Each computed probability in table B indicates the likelihood that both events occur simultaneously.

(a) min (g_{Ai}, g_{Bj})

min	10	12	14
2	2	2	2
4	4	4	4
6	6	6	6

(b) Multiplied probabilities (g_{Ai}, g_{Bj})

min	10	12	14
2	0.01166009785	0.08615711716	0.01166009785
4	0.08615711716	0.6366197722	0.08615711716
6	0.01166009785	0.08615711716	0.01166009785

Table 4.3: Merging value-probability pairs, summation of probabilities.

(a) min Intermediary result

Value	Probability
2	0.01166009785
2	0.08615711716
2	0.01166009785
4	0.08615711716
4	0.6366197722
4	0.08615711716
6	0.01166009785
6	0.08615711716
6	0.01166009785

(b) Aggregate MIN result, histogram H_C

Segment	Value	Probability
g_{C1}	2	0.1094773129
g_{C2}	4	0.8089340065
g_{C3}	6	0.1094773129

which were discussed in section 4.1.

Figure 4.3 shows us a Gaussian distribution A with parameters mean 4 and variance 1 and a Gaussian distribution B with parameters mean 12 and a variance of 1. The two Gaussian distributions are independent. Discretizing the two distributions yields two separate histograms as depicted in figure 4.4(a) and figure 4.4(b), respectively. The Middle Riemann Sum method is used here for the approximation of the probability density functions. Table 4.1(a) depicts the value-probability pairs, g_{Ai}, which correspond with each segment, i, in histogram H_A. This histogram has been derived from distribution A. Table 4.1(b) denotes those properties for each segment in histogram H_B. The value in each value-probability pair, g_{Ai} and g_{Bi}, refers to the middle of each segment. It is required for the other stages in the aggregation process that the step sizes of the histograms are the same. The attentive reader will notice that the probabilities or the areas of the columns in the histogram do not add up to one. The error is caused by the Middle Riemann Sum method which is used as a technique for approximating the continuous distribution. There are two classes of methods for sampling a continuous distribution. In one class, standard methods like Riemann sums and the Trapezoidal rule are used. The sampling methods in the other class ensure that the sum of approximated probabilities equals one. An example of these sampling methods is to use the Middle Riemann Sum method in conjunction with normalization of the approximated probabilities in such way that their sum equals one. However, the implication of using such method, is that the associated value moves. For this reason, we will use the Middle Riemann Sum method. The consequence is that the approximation errors will be propagated during the aggregation process; the errors will affect the final aggregation result.

A histogram can be considered as a sequence containing value-probability pairs. If we consider sequence S_A to be the sequence containing all the value-probability pairs corresponding to each segment in histogram H_A and sequence S_B corresponds similarly to histogram H_B, we can take the cartesian product of those two sequences. The result of this operation is: $R = [[g_{A1}, g_{B1}], [g_{A1}, g_{B2}], [g_{A1}, g_{B3}], [g_{A2}, g_{B1}], [g_{A2}, g_{B2}], [g_{A2}, g_{B3}], [g_{A3}, g_{B1}], [g_{A3}, g_{B2}], [g_{A3}, g_{B3}]]$. It is a sequence of sequences. In terms of possible world semantics, the result of the cartesian product operation is a collection of possible worlds. A possible world corresponds to a sequence $[[g_{Ai}, g_{Bj}]]$.

Now, we can apply selection function over the values from each pair, g, in each sequence, $[[g_{Ai}, g_{Bj}]]$. The selection function is one of the following functions: min, max, avg, sum, product. Applying the min over each value from each pair in the sequence $[g_{A1}, g_{B1}]$ means that $\min(2, 10)$ will be executed, yielding a result of 2. Moreover, the probabilities from each pair in the sequence $[[g_{Ai}, g_{Bj}]]$ are multiplied resulting in the probability that the events corresponding to the segment g_{Ai} are occurring and the events corresponding to the segment g_{Bj} are occurring at the same time. Multiplying the probabilities from each pair in the sequence $[g_{A1}, g_{B1}]$ is equivalent with $0.1079819330 * 0.1079819330 = 0.01166009785$. Combining the result of the selection function with the result of the multiplication of the original probabilities yields a new value-probability pair. If we apply this algorithm on each element in the sequence R, we get a number of value-probability pairs. Table 4.3(a) shows us the result of applying this algorithm on histograms A and B, with min as selection function. This part of the aggregation process is also called the *selection phase* since one value is computed or *selected* using a *selection function* from each sequence, $[[g_{Ai}, g_{Bj}]]$, in the sequence R. We showed that one sequence $[[g_{Ai}, g_{Bj}]]$ corresponds to one possible world, the sequence R corresponds to the collection of possible worlds. So, the selection function is evaluated in each possible world that is part of the collection of possible worlds. The results of the selection in each of the worlds is collected yielding a number of possibilities.

As we can see in table 4.3(a), the values in the value-probability pairs may overlap. For this reason, we introduce an additional step. We merge the value-probability pairs that contain the same value resulting in a set of unique pairs. In the *merging phase*, a summation is performed over the probabilities that correspond with the same values. This is because merged value-probability pairs should have a higher weight in the resulting distribution. In terms of semantics, the possibilities having the same value, are taken together.

Table 4.3(b) shows us the distribution resulting from the application of the aggregate AMIN function on histograms A and B. The result *should* be equal to histogram H_A because the values corresponding to the segments in histogram H_A are smaller than the values of the segments in histogram H_B. The cause for the error between the resulting histogram and the original histogram H_A is that the areas of the columns in histogram H_A do not add up to one. The error occurred during the sampling of continuous distributions is propagated in the computation and therefore affects the result, as has been explained before. As shown in table 4.3(b), the resulting distribution is approximated with a histogram having a step size of 2. The value corresponding to each segment lies in the middle of the segment.

The aggregation of continuous distributions results into one continuous distribution. Figure 4.5 shows an overview of the aggregation process. The aggregation process is based on possible world semantics. To summarize, the cartesian product of all possibilities residing in each distribution is taken. The result is a collection of possible worlds. The next step involves the selection phase. In this step, a possibility is selected from each possible world using a selection function. The selected possibility is associated with a probability that is equal to the multiplication of probabilities that are available in the corresponding possible world. The results, from selecting a possibility in each possible world, are collected. The next step involves the merging of possibilities having the same value. In this merging step, probabilities are summed. These last two steps are the actual aggregation of possibilities. The result of this aggregation process is a number of combined possibilities having a value and an associated probability. The result can be approximated with a histogram. As explained before, a very fine-grained histogram on a relative large interval approaches a continuous distribution. When Gaussian distributions are aggregated, the aggregation process may produce a Gaus-

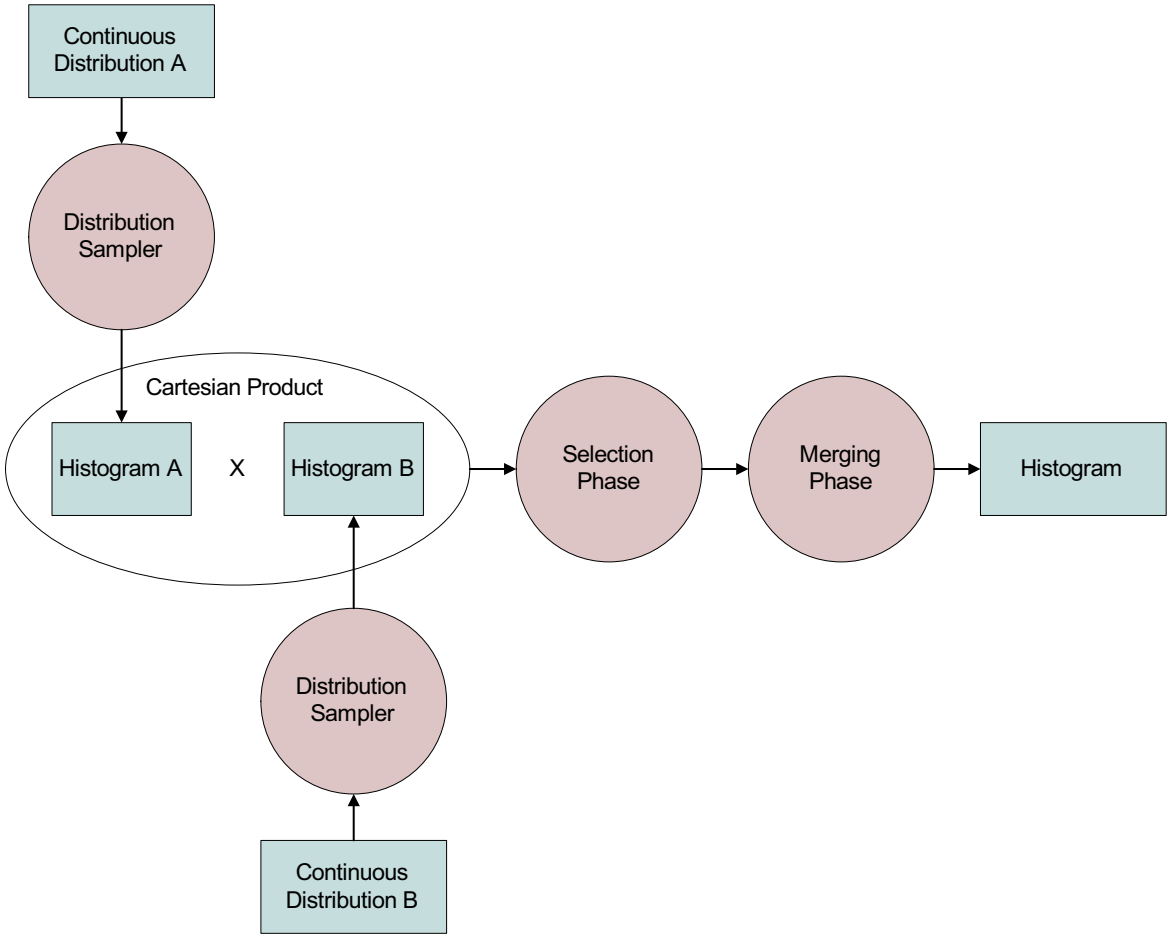


Figure 4.5: *An overview of the aggregation process.*

sian distribution which is approximated by a histogram. An example is the application of the ASUM or the AAVG operators with two Gaussian distributions, the result is one Gaussian distribution.

A detailed description of the aggregation operators is given by the algorithms in figure 4.3 and in figure 4.3. The algorithms are written in pseudocode. In the first figure, the sampling and the cartesian product functions are described. The cartesian product function uses several helper functions. The algorithm in the second figure describes the selection and merging phase. The selection function is responsible for computing the AMIN, AMAX, ASUM or the AAVG from the continuous distributions. The merge function merges value-probability pairs that contain the same value.

4.3.1 Scenario

In the previous section we introduced several aggregate operators that are capable of aggregating multiple continuous distributions yielding one single continuous distribution. In this section we will motivate the need for these operators with the help of a scenario.

The ”Brouwerij” is the name of a large brewery located in Enschede, The Netherlands. This

```
sample(distribution) : [(value, pdf(value))]
begin
        delta := 0.1; start := getStartOfDistribution(distribution); end := getEndOfDistribution(distribution)
        result := []
        let cnt := start
        while (cnt < end)
                result := result ∪ (cnt, pdf(cnt) ∗ delta)
                cnt := cnt + delta
        return result
end

cartesianProduct(array[] H) : [[(value, probability)]]
begin
        A := H[1]
        B := H[2]
        position := 3
        result := cartesianSimpleProduct(A, B)
        return cartesianRecursive(H, position, result)
end

cartesianRecursive(H, position, result) : [[(value, probability)]]
begin
if (position ¡ count(H)) then
        res := set:cartesianSimpleProduct(result, H[position])
        pos := position + 1
        return cartesianRecursive(H, pos, res)
else
        res := cartesianSimpleProduct(result, H[position])
        return res
end

cartesianSimpleProduct(array[] A, array[] B) : [[(value, probability)]]
begin
        result := []
        for each a ∈ A
                PW := [a]
                for each b ∈ B
                        PW := PW ∪ b
                result := result ∪ PW
        return result
end
```

Figure 4.6: Algorithms for sampling and the cartesian product.

brewery produces beer under different brands, its target is the whole European market. The ”Brouwerij” wants to control the quality of the production of beer continuously. Moreover, the goals of the management of the brewery is to gain competitive advantage. For that reason, the management decided to cut costs and to improve the operational management of the production process. The brewery does have an enterprise resource planning system. Among other things, this system helps to improve operational management and to keep the quality of beer at the same constant level.

The factory does have two production lines for bottling bottles of beer. Both production lines have a certain capacity a day, this capacity is Gaussian distributed. Production line one is distributed with parameters mean 10000 bottles and variance 1000 bottles, production line two is distributed with parameters mean 20000 and variance 2000. The management wants to know the total bottling capacity. In this situation, the ASUM operator can be used to sum both distributions.

```
select(selectiontype, C) : [(value, probability)]
begin
        mins := []; maxs := []; sums := []; avgs := []
        for each seq ∈ C
                values := []
                probability := 1
                for each pair ∈ seq
                        values := values ∪ firstElement(pair)
                        probability := probability * secondElement(pair)
                        mi := MIN(values); mins := mins ∪ (mi, probability)
                        ma := MAX(values); maxs := maxs ∪ (ma, probability)
                        su := SUM(values); sums := sums ∪ (su, probability)
                        av := AVG(values); avgs := avgs ∪ (av, probability)
        if (selectiontype == "MIN") then return mins
        else if (selectiontype == "MAX") then return maxs
        else if (selectiontype == "SUM") then return sums
        else if (selectiontype == "AVG") then return avgs
        else []
end

merge(P) : [(value, probability)]
begin
        result := []
        for each pair ∈ P
                value := firstElement(pair)
                probability := −1
                for each pair_res ∈ result
                        value_res := firstElement(pair_res)
                        if (value==value_res) then
                          prob_p := secondElement(pair)
                          prob_res := secondElement(pair_res)
                          probability := prop_p + prob_res
                          updateSecondElement(pair_res, probability)
                        else
                if (probability == −1) then
                        result := result ∪ [(value, secondElement(pair))]
                else
        return result
end
```

Figure 4.7: Algorithms for selecting and merging.

The filling process consists of different stages. One stage of the process is filling the bottle, the next step is pushing the caps on the bottle. The filling machine fills a beer bottle imprecisely. The amount of beer in a bottle is Gaussian distributed with an expected value of 33 centiliters and a variance of 5 centiliters. While pushing caps on a bottle, some bottles get broken due to too much pressure of the pushing machine or damaged glass. The beer inside broken bottles will be lost in the production line. The daily amount of damaged bottles in the bottling process is Gaussian distributed with parameters mean 50 bottles and a variance of 10 bottles. The management wants to know the amount of lost beer in the bottling process. This can be computed by applying the PRODUCT operator over the distribution denoting the amount of beer in a bottle and over the distribution representing the amount of broken bottles.

A study towards the operational performance of the bottling line makes clear that the capacity of the bottling line is heavily dependent on which team of employees is on duty. The management would like to fire bad performing teams, while good performing teams may

Table 4.4: The resulting values of the APRODUCT aggregate over (g_{Ai}, g_{Bj})

product	10	12	14
2	20	24	28
4	40	48	56
6	60	72	84

get a bonus. During one month, the bottling capacity is measured for each team that was in charge. The result is a set of Gaussian distributions, each distribution corresponds to the performance of one team in a particular month. The AMIN operator is executed over these distribution resulting in the distribution that indicates the likelihood that a team performs badly. Similarly, the AMAX operator is executed. The management introduces a general rule: when the production of one team in the next month is less than the expected value of the minimum performance in the previous month, and when this happens two times, the team as a whole is fired. Bonuses are provided similarly.

Besides, the production facilities, the factory does also host a laboratory which is monitoring the quality of beer continuously by taking samples and do experiments on them. One of the experiments is the measurement of the alcohol percentage. Due to impreciseness of instruments, the measurements of the alcohol percentage have a Gaussian distribution. The laboratory would like to aggregate the measurement results of the past month. One approach for doing this, is to use the AAVG operator to compute the average distribution.

4.3.2 A special case: the APRODUCT aggregate operation

In section 4.3 we presented an approach for aggregating continuous distributions. One type of aggregation is the APRODUCT aggregate, the need for this type was motivated in the previous section. In contrast to other aggregate operations, the result of the APRODUCT aggregate cannot be approximated with a histogram.

Consider table 4.2(a), when we apply the APRODUCT operator instead of the MIN function, we will get a distribution containing the values that are shown in table 4.4. There is no fixed step size, the step size varies between 4 and 12. This result cannot be represented by a histogram since a histogram has, by definition, a fixed segment size. This is still an open issue.

Chapter 5

Prototype

While the previous chapters focused more on the theoretical aspects of supporting continuous uncertainty by semi-structured databases, this chapter will describe the practical issues dealt with during the implementation of a prototype supporting continuous uncertainty. The most important goal of this prototype is to show the ability of managing continuous uncertain data. In this chapter we will discuss the development of a proof of concept supporting the model and concepts presented in the previous chapters. This chapter is structured like any other software design document; it contains sections about the requirements, design and implementation.

5.1 Requirements

Since our study focused primarily on a data model and a query language for continuous uncertain semi-structured data, these are the most important aspects of our proof of concept. We distinguish the following requirements:

1. Probabilistic XML
   In chapter 3 we explained that our model for continuous uncertainty is based on Probabilistic XML [12, 23]. For that reason, the data format used in the prototype should resemble the continuous representation which we proposed in chapter 3. We need to find a way to encode probability density functions, which are inherently related to continuously uncertainty, into Probabilistic XML documents. Moreover, the implementation of the query language should be able to work on these specific versions of Probabilistic XML documents.

2. Continuous distributions
   The common continuous probability distributions like beta, gamma, gaussian and the uniform distributions can be stored in XML using the data model that was proposed in chapter 3. We require that the proof of concept has support for at least one of the following continuous distributions: beta, gamma, gaussian or the uniform distribution. Extending the prototype in such way that it will support other well-known continuous probability distributions is straightforward then. Furthermore, the prototype should support storing and querying non-standard continuous probability distributions (e.g distributions that have been floored).

3. Probabilistic query constructs
   One of the major issues we would like to evaluate is the query language as proposed in 4. The prototype should have support for all the operations that were discussed in the previous chapter; operators like mean, variance and the aggregate functions. The implementation should have support for using operators in predicate queries.

4. Symbolic representation
   Although the approximations of continuous distributions can be represented by histograms as was shown in chapter 3, the only way to store this data without loss of information is to store the symbolic form of a probability density function itself (and its parameters). The implementation should have support for encoding various kinds of probability density functions in the symbolic form in probabilistic XML documents.

5. Historical dependencies
   As discussed above, the symbolic representation ensures that no information will be lost when managing continuous uncertain data. The ideal situation is that the symbolic representation captures the operations that were performed on the base probability density functions. Then, the symbolic representation acts as a log that keeps track of the history of a continuous stochastic variable.

6. Histograms
   The symbolic representation is able to represent any kind of distribution in an accurate way when it stores the distributions as a sequence of operations that have been performed on base probability density functions. Consider the application of the FLOOR operator on a standard Gaussian distribution. This results in a non-standard distribution, the symbolic representation would represent this as a floor function applied on a gaussian probability density function. The exact formula is, in most situations, unknown or is too complicated to compute. Suppose we would like to compute a probability, the mean or variance of this non-standard distribution, then we have to compute the area under the graph of the resulting probability density function dynamically at run-time. This is too time-consuming. For this reason, the prototype should have support for histograms which can capture non-standard distributions such as distributions that have been floored or aggregated. The symbolic representation has to be stored along with the histogram representation.

7. Understandable presentation
   The query language consists of complex constructs for querying continuous uncertain data. The presentation of the query results to the end-user should be understandable so that non-advanced end-users are able to work with this prototype.

8. XQuery
   XPath/XQuery is the *de facto* standard for querying XML documents. The prototype should have support for querying Probabilistic XML documents, containing continuous uncertain data, using the XPath/XQuery language. So it should be possible to use the probabilistic query constructs, that were presented in chapter 4, in XPath/XQuery queries

9. Extensibility/Scalability (non-functional)
   The piece of software we are developing, is a research prototype. The prototype maybe

```
<rooms>
    <room>
     <number>1</number>
        <prob>
            <poss type="gaussian(15,3)" approximation="true"
                approximation-type="middle-riemann-sum" left="6"
                right="24" delta="2" heights="(3.157722291e-7,0.00006540503306,
                0.003570993807,0.05139344328,0.1949696557,0.1949696557,
                0.1949696557, 0.003570993807,0.00006540503306,3.157722291e-7)">
                    <temperature/>
            </poss>
        </prob>
    </room>
    <room>
     <number>2</number>
        <prob>
            <poss type="gaussian(23,1)" approximation="true"
                approximation-type="middle-riemann-sum" left="20"
                right="26" delta="2" heights="(0.004431848411, 0.2419707244, 0.2419707244,
                0.2419707244)">
                    <temperature/>
            </poss>
        </prob>
    </room>
</rooms>
```

Figure 5.1: *A Probabilistic XML document containing continuous uncertain data.*

extended in a later stage with support for other continuous distributions or other concepts/functionalities to be discussed in section 7.3.

5.2 Design

As mentioned before, the aim of the prototype is to manage continuous uncertain data. The prototype allows the end-user to query continuous uncertain data using a query language. Query languages are data-oriented languages, a language should be able to process a database having a certain schema. Data formats or structures form an important aspect of our prototype. In this section we will discuss the structure of XML documents that are supported by the prototype.

5.2.1 Probabilistic XML (continuous representation)

An example of a Probabilistic XML document containing continuous uncertainty is provided in Figure 5.1. The probabilistic XML document concerns the temperatures of two rooms that are located in a home for elderly people. The room temperature has been measured by temperature sensors resulting in imprecision. The temperature can be described by Gaussian distributions. The continuous distributions are encoded in the document with the help of separate prob- and poss-nodes, the children of a prob-node enumerate the possible alternatives

and a poss-node encodes a single probability or a probability density function. The poss-node holds both the symbolic as well as the histogram representation. The gaussian distribution is stored by the poss-node as if it is a value of the type-attribute of the poss-node. The value of the type-attribute expresses the gaussian distribution as a function which has two parameters: the mean and the variance. The type-attribute actually models the symbolic representation of the continuous distribution, for instance: a floor operator performed on the `gaussian(15,3)` on the interval [20, 24] would be represented as `type=floor(gaussian(15,3),[20,24]))`. The symbolic representation is a nested chain of probabilistic operations. The approximation of the symbolic representation is represented by a histogram, the left and right boundaries of the interval of the histogram, the step size and the heights corresponding to each segment, are all stored as parameters of attributes within the poss-node. It is often the case that a continuous distribution is not bounded by an interval, for instance the curve of the Gaussian distribution does never reach the x-axis. It is practically impossible to store the whole histogram corresponding to a continuous distribution. A histogram is always specified with an interval and/or the size of the histogram representation will become infinite. For this reason, the prototype stores only that part of a continuous distribution where the probability mass is concentrated. In case of a Gaussian distribution, it means that the histogram stores the pairs of the value and the corresponding output of the probability density function on the interval [(mean - 3 * standard deviation), (mean + 3 * standard deviation)]. This interval reflects 99.73 percent of the probability mass.

The advantage of using the Probabilistic XML data format is that there is a one-to-one mapping possible between the theoretical data model presented in chapter 3 and the XML document as presented in figure 5.1. However, as we can see, some 'hacks' were necessary to store the parameters of the continuous distribution and the heights of the histogram. The information is formatted as string-data and saved as values of different attributes. During query processing, all this string-data have to be parsed in an appropriate way. Another issue is that there is no definition/specification language available for specifying the data format or schema of these attribute values. A better approach is to store the symbolic representation and the histogram representation in separate subtrees. We will describe this approach in the next section.

5.2.2 Continuous Uncertain XML

Figure 5.2 shows us an XML document representing a continuous distribution. The distribution is represented by a symbolic representation and a histogram representation. The symbolic and the histogram representations are both stored as separate child-nodes of the distribution-node. The probability density function is modeled as a separate XML element which is a child of the symbolic-representation-node, the parameters of the distribution are represented by attributes. The height of each column in the histogram is stored in an y-node. In the previous section we mentioned, that it is impossible to represent a continuous distribution by a histogram without loss of information; the histogram representation stores only that part of the continuous distribution where the probability mass is significant. This means that, in case of a Gaussian distribution, only the interval of the mean plus/minus 3 times the standard deviation is stored by the histogram representation. As mentioned before, the symbolic representation keeps track of the actions that were applied on base continuous distributions. An advantage of the approach presented in figure 5.2, is that it uses the expressive power of XML to model nested hierarchies for the symbolic representation. The

```
<distribution>
    <symbolic><gaussian variance="3" mean="15"/></symbolic>
    <histogram left="6" right="24" delta="2">
        <y>3.157722291e-7</y>
        <y>0.00006540503306</y>
        <y>0.003570993807</y>
        <y>0.05139344328</y>
        <y>0.1949696557</y>
        <y>0.1949696557</y>
        <y>0.05139344328</y>
        <y>0.003570993807</y>
        <y>0.00006540503306</y>
        <y>3.157722291e-7</y>
    </histogram>
</distribution>
```

Figure 5.2: *An XML document containing a continuous distribution.*

```
<distribution>
    <symbolic>
        <AMIN>
            <distribution>
                <symbolic><gaussian variance="3" mean="15"/></symbolic>
            </distribution>
            <distribution>
                <symbolic><gaussian variance="1" mean="23"/></symbolic>
            </distribution>
        </AMIN>
    </symbolic>
</distribution>
```

Figure 5.3: *The symbolic representation of the AMIN operation applied on two continuous distributions.*

result of applying the AMIN function over two Gaussian distributions can be presented by the symbolic representation as is shown in figure 5.3.

Although the Continuous Uncertain XML approach differs from the theoretical data model presented in chapter 3, the Continuous Uncertain XML representation is fully compatible with this data model. The only difference with the Continuous representation is that the continuous distribution is modeled as a separate subtree below the poss-node, with the distribution-node being the root of the subtree.

Another major advantage of this approach is that the format of these kind of XML documents can be specified using XMLSchema or a DTD. This allows the use of a validator which checks whether an instantiation complies to the schema of Continuous Uncertain XML. The schema of Continuous Uncertain XML is given in appendix A.

5.2.3 Query processing

The prototype will support the query constructs that were discussed in chapter 4, except the APRODUCT operator. The query constructs can be used in compositions of XQuery queries. Since the prototype supports both standard as well as non-standard distributions, histograms are needed to capture the distributions. We mentioned in the previous section that the histogram representation is stored along with the symbolic representation because it is too time-consuming to generate a histogram out of a symbolic representation at query time. It is possible to use the symbolic representation when querying a standard distribution. However, for reasons of simplicity, we decided that the query operators will always work directly on the histogram representation of a continuous distribution instead of making a distinction between standard and non-standard distributions and routing the query to the appropriate representation when executing a query. Thus, the query operators operate directly on histograms. Because of the inherent inaccuracy of histograms, the answers returned by these query operators are at most approximations.

5.2.4 Continuous distributions

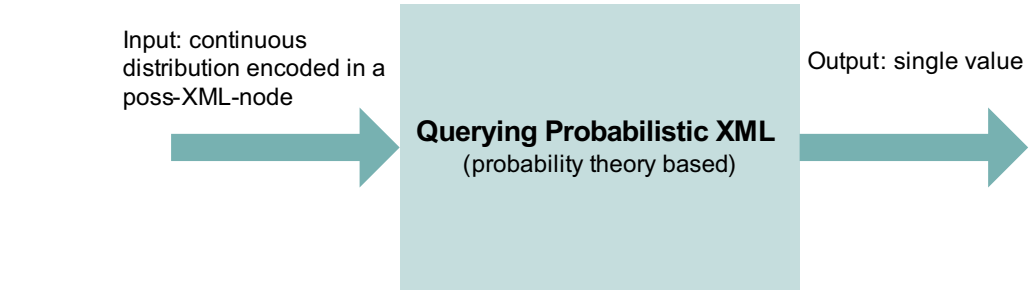
We posed the requirement that the prototype should have support for at least one continuous distribution. The prototype will support the storage of Gaussian distributions by the symbolic representation since this is a commonly used continuous distribution. Next, the prototype will contain functions which are capable of transforming the symbolic representation of a Gaussian distribution to a histogram representation. In addition, the symbolic representation and the implementation of the query language will support the use the floor operators over Gaussian distributions.

5.3 Architecture

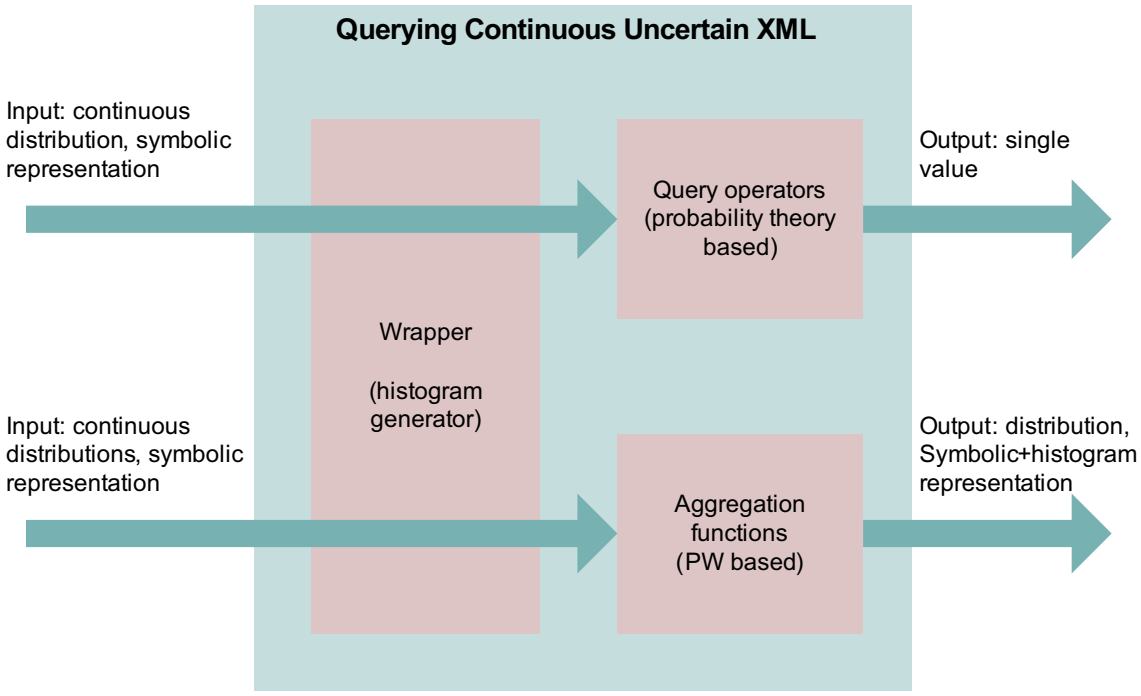
The implementation of the prototype supports both the Probabilistic XML and the Continuous Uncertain XML data formats. Figure 5.4 contains two diagrams depicting the architecture of the prototype. The prototype consists of two parts: one component is responsible for query operators that are able to process Continuous Uncertain XML, the other component of the prototype consists of operators that are capable in processing Probabilistic XML. Although the prototype consists of two separate components, the name and the semantical meaning of the query operators in both components are equal.

The prototype is implemented using the XQuery language and it runs on MonetDB/XQuery. The XQuery code is organized equivalently with the architecture: there is a set of modules (namespaces) which implement the features for querying Probabilistic XML and there is a set of modules which implement querying Continuous Uncertain XML. Each module is a collection of XQuery functions, a module implements at most one operator of the query language for continuous uncertain data.

As shown in figure 5.4, the component responsible for querying Continuous Uncertain XML, contains a wrapper component that generates histograms from a symbolic representation. The histogram generator allows us to invoke the query operators of the prototype with a symbolic representation. Then, the prototype generates the histogram corresponding to the symbolic representation and passes it to the appropriate (low-level) query operator.



(a) Probabilistic XML



(b) Continuous Uncertain XML

Figure 5.4: *The architecture of the prototype.*

The result of an aggregate query operator is a continuous distribution represented by a histogram. Due to the complex way of representing histograms in Probabilistic XML, we decided to leave out the functionality in the prototype for performing aggregate operations on Probabilistic XML. The support for the Probabilistic XML data format is limited to query operators like P, vmin, vmax, mean and variance.

5.4 Implementation details

The XQuery code which implements the prototype is divided into several XQuery modules. Each module contains several functions. In this section, we will discuss several aspects of the implementation of the Continuous Uncertain XML representation such as how the symbolic representation is transformed to histograms, how histograms are processed by the query

operators and how the aggregation of distributions is implemented.

5.4.1 Symbolic to histogram

In order to transform a symbolic representation to a histogram representation, a recursive function is required. This recursive function traverses the nested chain of operations that have been performed on the base probability density function. While traversing the nested chain of operations, each operation is applied on the operation that has been performed before resulting in an updated histogram. Thus, the chain is scanned backward and the updated histograms are propagated forward finally resulting in a histogram which corresponds with the symbolic representation. The XQuery implementation is shown below.

```
declare function symbolic:createHistogramFromSymbolic($distribution as node(), $delta as
    xs:double) as node() {
let $op_or_pdf := exactly-one($distribution/symbolic/child::node())
let $name := local-name($op_or_pdf)
return
  if($name eq "gaussian") then
    <distribution><symbolic>{$op_or_pdf}</symbolic>
    {symbolic:createHistogramFromSymbolicGaussian($op_or_pdf, $delta)}
    </distribution>
  else if($name eq "A_MIN") then
    let $agg_childs := $op_or_pdf/child::node()
    let $distributions := for $dist in $agg_childs return
                          <distribution><symbolic>{$dist/symbolic/child::node()}</symbolic>
                          {symbolic:createHistogramFromSymbolic($dist, $delta)/histogram}
                          </distribution>
    let $result := exactly-one(a_min_ed:A_MIN_ED($distributions))
    return $result
  else if($name eq "A_MAX") then
    let $agg_childs := $op_or_pdf/child::node()
    let $distributions := for $dist in $agg_childs return
                          <distribution><symbolic>{$dist/symbolic/child::node()}</symbolic>
                          {symbolic:createHistogramFromSymbolic($dist, $delta)/histogram}
                          </distribution>
    let $result := exactly-one(a_max_ed:A_MAX_ED($distributions))
    return $result
  else if($name eq "A_AVG") then
    let $agg_childs := $op_or_pdf/child::node()
    let $distributions := for $dist in $agg_childs return
                          <distribution><symbolic>{$dist/symbolic/child::node()}</symbolic>
                          {symbolic:createHistogramFromSymbolic($dist, $delta)/histogram}
                          </distribution>
    let $result := exactly-one(a_avg_ed:A_AVG_ED($distributions))
    return $result
  else if($name eq "A_SUM") then
    let $agg_childs := $op_or_pdf/child::node()
    let $distributions := for $dist in $agg_childs return
                          <distribution><symbolic>{$dist/symbolic/child::node()}</symbolic>
                          {symbolic:createHistogramFromSymbolic($dist, $delta)/histogram}
                          </distribution>
    let $result := exactly-one(a_sum_ed:A_SUM_ED($distributions))
    return $result
  else error("An undefined error occurred while parsing the symbolic representation.")
};
```

Listing 5.1: *The implementation of symbolic-to-histogram transformer.*

When this recursive algorithm reaches the base probability density function, in this case the Gaussian distribution, the Gaussian distribution is sampled yielding a histogram. The XQuery code implementing this function is shown below:

```
declare function symbolic:createHistogramFromSymbolicGaussian($gaussian as node(), $delta
    as xs:double) as node() {
let $sigma_cnt := 3
let $factor := 1 div $delta
let $mean := (exactly-one($gaussian/@mean)) cast as xs:double
let $variance := (exactly-one($gaussian/@variance)) cast as xs:double
let $sd := pf:sqrt($variance) cast as xs:double
let $left := gaussian:getStartOfDistribution($mean, $sd, $delta, $sigma_cnt)
let $right := gaussian:getEndOfDistribution($mean, $sd, $delta, $sigma_cnt)
let $start := ($left div $delta) cast as xs:integer
let $end := ($right div $delta) cast as xs:integer
return <histogram left="{$left}" right="{$right}" delta="{$delta}">
        {
        for $v at $index in ($start to $end)
          let $x := ($v*$delta)
          return <y>{gaussian:gaussian-density($x, $mean, $sd)}</y>
        }
        </histogram>
};
```

Listing 5.2: *The implementation of the histogram generator.*

We can see in the code that there are a lot of division and multiplications by delta. This is because the for-loops can only process integers. When we divide the boundaries of the histogram by the delta, we get start- and end-values that are round integers.

5.4.2 Histograms

The histogram representation is defined as a histogram-node holding the values for the left- and right-boundaries and the delta as values of attributes. The height of each column in the histogram is represented by a y-node which is a child of the histogram-node. Each column in the histogram corresponds to a value and an height. The area of the column (probability) is computed by multiplying the height with the delta of the histogram. Each column in a histogram can be identified by a unique number. We implemented functions for retrieving the value given the column-identifier and vice versa, retrieving the height given the column-identifier, computing the probability given the identifier and so on. The identifier corresponds to the relative location of the y-node under the histogram-node.

The function below returns the height of the column, given the column-identifier (index).

```
declare function hist:getHeightByIndex($histogram as node(), $index as xs:integer) as
    xs:double {
  data(exactly-one($histogram/child::node()[$index])) cast as xs:double
};
```

Listing 5.3: *This function returns the height of a column in the histogram.*

The function below returns the value which is associated with a column, given the column-identifier (index).

```
declare function hist:getValueByIndex($histogram as node(), $index as xs:integer) as
    xs:double {
  if( ($index le hist:getLastIndex($histogram)) and ($index ge 1) ) then
    let $delta := hist:getDeltaOfHistogram($histogram)
    let $left := hist:getLeftOfHistogram($histogram)
    let $l := ($left div $delta) cast as xs:integer
    let $v := (($l + $index) - 1)
    let $value := ($v * $delta) cast as xs:double
    return $value
  else
```

```
10      error("The specified index is lies out of the interval for which the histogram was
            specified.")
11  };
```

Listing 5.4: *This function returns the value which is associated with a column in the histogram.*

5.4.3 Simple probabilistic functions: Pr, mean, variance

The implementation of operators like Pr, mean and variance process (parts of) the histograms that represent continuous distributions. The Pr function has to determine the left- and right-boundaries which were specified by the end-user when composing the query. These boundaries indicate the interval of the continuous distribution for which it has to compute the probability. The Pr function is shown below. The implementations of the mean and variance operations are similar to the implementation of Pr.

```
1  declare function prob:P($distribution as node(), $f_interval as xs:string) as xs:double {
2    let $histogram := exactly-one($distribution/histogram[1])
3    let $left_index := parsers:getLeftBorderIndex($histogram, $f_interval)
4    let $right_index := parsers:getRightBorderIndex($histogram, $f_interval)
5    let $prob := sum(for $y at $index in $histogram/y
6                     return if(($index gt $left_index) and ($index lt $right_index)) then
7                               hist:getProbByIndex($histogram, $index)
8                             else if(($index = $left_index) or ($index = $right_index))
                                  then
9                               hist:getProbByIndex($histogram, $index) div 2
10                            else()
11                 )
12   return $prob
13 };
```

Listing 5.5: *This function computes the probability that corresponds to the given interval and continuous distribution.*

5.4.4 Aggregation operators

The implementation of the aggregation operators resulted in a complex piece of XQuery code. We faced two main problems during the implementation of the aggregation operators:

1. XQuery does not support the use of sequences within sequences (nesting)
   A sequence within a sequence will be flattened.

2. There is no out-of-the-box support for data objects that can represent x-y pairs
   This makes the use of value-probability pairs, as it was used in the algorithms in figure 4.3 and figure 4.3, difficult.

A solution to the first problem was using XML nodes to encode sequences. Each sequence is encoded in a separate seq-node. Each element in a sequence is encoded in a separate e-element which itself is a child of the seq-node. This approach allowed us to represent a histogram as if it is a sequence. The disadvantage of this approach is that a lot of intermediary XML nodes have to be created within the XML database management system. MonetDB/XQuery performs poorly when intermediary XML nodes have to be generated.

The second problems was tackled by using an identifier structure for retrieving the corresponding value and probability of a column in a histogram. This identifier is based on

the order of columns that are representing a histogram (left to right). By keeping track of the order of histograms during query processing in conjunction with the column identifier, we are able to retrieve the (value, probability) pair at any time.

When we combine the solutions that are explained above, a set of histograms can be represented as if it is a sequence of sequences. Each sequence in the sequence represents a histogram, each element in the deepest sequence contains an identifier which maps to a (value, probability)-pair of a certain histogram. The location of the histogram-sequence in the sequence of histograms, determines the histogram to be used for retrieving the (value, probability)-pair. The XQuery code of the AAVG operator is shown below.

```
declare function a_avg_ed:A_AVG_ED($distributions as node()*) as node()* {
    let $histograms := for $dist in $distributions return $dist/histogram
    let $left := a_avg_ed:getAvgOfLefts($histograms)
    let $right := a_avg_ed:getAvgOfRights($histograms)
    let $delta := hist:getDeltaOfHistogram(exactly-one($histograms[1]))
    let $count_hist := count($histograms)
    let $new_delta := ($delta div $count_hist) cast as xs:double
    let $start := ($left div $new_delta) cast as xs:integer
    let $end := ($right div $new_delta) cast as xs:integer
    let $possible_combinations := agg:getPossCombsOfIndices($histograms)
    let $y_values := for $item at $index_in_hist in ($start to $end)
                        let $x_value := ($item * $new_delta) cast as xs:double
                        let $y_value :=
                            (a_avg_ed:getProbabilityPWAVG($possible_combinations,
                            $histograms, $x_value) div $delta) cast as xs:double
                        return <y>{$y_value}</y>
    let $histogram := <histogram left="{$left}" right="{$right}"
        delta="{$new_delta}">{$y_values}</histogram>
    let $symbolic := <symbolic><A_AVG>{for $dist in $distributions return
                                        <distribution>{$dist/symbolic}</distribution>
                                    }
                     </A_AVG></symbolic>
    let $distribution := <distribution>{$symbolic, $histogram}</distribution>
    return $distribution
};
```

Listing 5.6: *This function covers the whole process of aggregating multiple continuous distributions using the AVG aggregate.*

The function above calls the agg:getPossCombsOfIndices($histograms) function with an array of histograms as argument. The getPossCombsOfIndices-function transforms each histogram in a sequence of identifiers, each identifier identifies a (value,probability)-pair. The implementation of the getPossCombsOfIndices-function is shown below. As we can see, the function calls the cartesianProduct-function which computes the cartesianproduct of the sequences containing the identifiers.

```
declare function agg:getPossCombsOfIndices($histograms as node()*) as node()* {
  set:cartesianProduct(
  for $histogram at $i in $histograms
      return
          <seq>
          {
            (: let $delta := hist:getDeltaOfHistogram($histogram) :)
            let $end := hist:getLastIndex($histogram) cast as xs:integer
            return for $index in (1 to $end) return <e>{($index)}</e>
          }
          </seq>
  )
};
```

Listing 5.7: *This function computes the probability that corresponds to the given interval and continuous distribution.*

The function that is shown below, implements the *selection* phase and the *merging* phase of the aggregation algorithm. It first computes the averages over each possible combination of values, then it computes the associated probabilities. When there are two or more values that are the same, the probabilities of those corresponding values are summed.

```
declare function a_avg_ed:getProbabilityPWAVG($poss_combs_indices as node()*, $histograms
    as node()*, $x_value as xs:double) as xs:double
{
let $sets := (for $set in $poss_combs_indices
                let $elements := $set/e
                let $avgg := avg(
                  for $element at $hist_index in $elements
                      let $index_in_hist := data($element) cast as xs:integer
                      let $histogram := exactly-one($histograms[$hist_index])
                      let $value_data := hist:getValueByIndex($histogram, $index_in_hist)
                      return $value_data
                )
                return if($avgg = $x_value) then $set else()
              )

return sum(for $set in $sets
            let $elements := $set/e
            let $product :=
                  pf:product(for $element at $hist_index in $elements
                              let $index_in_hist := data($element) cast as xs:integer
                              let $histogram := exactly-one($histograms[$hist_index])
                              let $prob := hist:getProbByIndex($histogram, $index_in_hist)
                              return $prob)
            return $product
          )

};
```

Listing 5.8: *This function computes the probability that corresponds to the given interval and continuous distribution.*

The aggregate operators are all processing the histograms that are encoded in the Continuous Uncertain XML representation. A problem occurs when an aggregate operator aggregates two or more continuous distributions and the corresponding histograms have different step sizes. The algorithm presented in section 4.3 requires that slices of the continuous distributions have the same size. There are several ways to solve this issue. Our approach is first to check whether the histograms vary with respect to the step size. If that is the case, the histograms are re-generated from the symbolic representation with a step size that is defined as a fixed constant within the database management system.

5.5 Using the prototype

This chapter would not be complete without a section on how to use the prototype software. In this section we present several examples containing queries for continuous uncertain data. We discuss querying Probabilistic XML and querying the Continuous Uncertain XML representation, separately.

Probabilistic XML

The queries that are shown by the following examples, use the Probabilistic XML document depicted in figure 5.1. In this query, the variable `$d` points to that document.

Example 1:

```
$d/room[number = 1]/Pr(.,"temperature < 16")
```

The query above returns the probability that the room with roomnumber 1 has a temperature of less than 16 celcius.

As mentioned in the previous section, all the query operators for continuous uncertain data only process discretized versions of continuous uncertain data (histogram representation). A histogram stores y-coordinates (which are ordered), a fixed step size which denotes the distance between two y-coordinates and the interval for which the histogram is specified. The x-coordinates can be determined from the location of ordered y-coordinates, the step size and the interval. Posing queries on the histogram representation brings us to a problem related to specifying a correct interval in query operators like Pr. Suppose the histogram representation that corresponds to the temperature-attribute from example 1 has a step size of 0.5. What will happen when we pose a query like `$d/room[number = 1]/Pr(.,"temperature < 16.1")` and the histogram is specified on the interval [14,20]? In this example, the value 16.1 lies in the segment associated with interval [16,16.5]. We can choose to exclude or include this segment when computing the probability. Whatever option we choose, we get an inaccurate answer then. The problem is that the given interval cannot be covered completely by histogram segments. Our solution is to return an error when this situation happens. The solution is implemented as follows:

1. Divide the x-coordinate by the step size.
2. Take the round over the result from the previous step.
3. The result from the first step minus the result from the second step.
4. If the result from the previous step is greater than zero, raise an error.

Example 2:

```
$d//room/mean(.,"temperature")
```

The query above computes for each room in document `$d` an approximation of the expected temperature.

The histogram representation is stored in a poss-node which is a child of a prob-node. The prob-node is a child of the room-node. As we can see in the examples above, it is not necessary to specify the `prob/poss`-node structure in the queries, explicitly. In both examples, the query context is provided to the query function, Pr and mean, respectively. The query context (location) is provided to the query function by invoking the function with "." as first parameter. When the query functions know the query context, the query functions can navigate through the prob/poss-node structure and the histogram can be found.

Continuous Uncertain XML

The major difference between Continuous Uncertain XML and Probabilistic XML is that the continuous distribution is represented in a subtree below a poss-node. The rootnode of the subtree is a distribution-node which contains a symbolic-node, holding the symbolic representation, and a histogram-node, holding the histogram representation. In contrast to querying Probabilistic XML, the query functions for querying Continuous Uncertain XML cannot be invoked with the query context (location) as parameter. When querying continuous distributions that are represented in Continuous Uncertain XML, the query function has to be invoked with as argument the location of the distribution-node in the XML document.

Example 1:

```
Pr($distribution,"[15,24]")
Pr($distribution,"]15,24]")
Pr($distribution,"[inf,24]")
```

All the queries in the example above compute the approximated probability corresponding to the interval and the continuous distribution indicated by `$distribution`. The `$distribution` could be the distribution depicted in figure 5.2. The second parameter of the Pr function is used to specify the interval for which we would like to know the corresponding probability. The notation is similar to other notations for specifying intervals, ”]15” excludes the value 15 from the interval, with ”[15” the value 15 is included. The string ”inf” or ”INF” can be used to specify infinity. So ”]INF, 5]” means $]-\infty, 5]$.

Example 2:

```
ASUM($distribution1,$distribution2)
```

The query in the example above aggregates two continuous distributions using the ASUM operator. The result is in an aggregated continuous distribution.

Chapter 6

Evaluation

The previous chapter addressed the issues related to the realisation of a prototype that is capable of querying continuous uncertain data which is represented by histograms. A histogram represents a number of segments on a certain interval. The ratio between the number of segments and the length of the interval determines the following:

- Efficiency
  The more segments that have to be taken into account during query execution, the more memory will be consumed for storing temporarily execution data. Thus, it will take more processing time.

- Accurateness
  The more segments on a certain interval, the smaller the segments are. We expect that a smaller segment size means that the answers returned by queries will be more precise/accurate.

In the world of database management systems, it is common to use benchmarks to test the performance of a DBMS. In general, benchmarks are used to assess the relative performance characteristics of software. A database benchmark is a prescribed set of queries and data that can be run against a DBMS. Benchmarking is helpful in making a fair comparison between database management systems by measuring the time of performing the same tests on different database management systems. These benchmarks are also applicable to assess the performance of a single database system, to determine the improvement in performance of e.g. a renewed implementation of a query operator. However, benchmarks for database management systems holding continuous uncertain semi-structured data do not exist as far as we know. For that reason, we have to setup new benchmark scenarios.

In this chapter we will quantify the relationship between the performance indicators efficiency and accurateness by conducting some experiments. We will discuss the set-up of the experiments, the results and we will analyse them.

6.1 Experiments

When we examine the complexity of the different query operators specified in chapter 4, the value-based operators like Pr, mean, variance differ from the aggregate operators like AMIN, AMAX, ASUM. The complexity of value-based operators is linear where the algorithm of

```
let $variance := 4
let $delta := 0.1

let $dist := <distribution><symbolic>
                <gaussian mean="0" variance="{$variance}"/>
             </symbolic></distribution>
let $dist_hist := symbolic:createHistogramFromSymbolic($dist, $delta)

let $intervalleft := concat('[', 0)
let $intervalright := concat('inf', ']')
let $intervalmiddle := concat($intervalleft, ', ')
let $intervalstring := concat($intervalmiddle, $intervalright)

let $prob_est := prob:P($dist_hist, $intervalstring)

return
<result>
  <inaccuracy>{abs(0.5 - $prob_est)}</inaccuracy>
</result>
```

Figure 6.1: *XQuery code used for measuring the accuracy and performance of the probability operator.*

aggregate operators is at least quadratic. The query operators can be grouped in value-based and aggregate operators based on the complexity. As the complexity is different, we have to setup different benchmarks for the two groups.

6.1.1 Value-based operators

The complexity of all value-based operators is the same, so it is not necessary to assess the performance of each individual query operator. It is interesting to examine how parameters like variance and the segment size influence the execution time and the accuracy when querying one continuous distribution using a value-based operator. Next, it is interesting to examine the effects on the response time of executing a value-based operator over several continuous distributions. This relationship, between the number of continuous distributions to be queried and the response time, describes the correlation between the size of a continuous uncertain XML document and the time required to query the document.

In the first experiment, we will examine how factors like variance and the delta influence the accuracy of the answer and efficiency of query execution. The experiment contains several tests where variance and the delta are the variables. In each test, a Gaussian distribution is generated and then the test computes the probability on the interval [*mean*, inf] using the Pr operator. This probability is theoretically equal to 0.5. The error can be measured by taking the absolute value from 0.5 minus the value computed by the Pr operator. So the deviation between the estimated answer and the exact answer is the error. Each test assesses the efficiency by measuring the query execution time. Each test is ran three times, so that execution times can be averaged. In the first test, the segment size of the histogram is the variable, the parameters for the Gaussian distribution are constants: mean=0, variance=4. The variance determines the length of the interval on which the histogram is defined. As

```
let $delta := 0.01
let $room_count := 90
let $variance := 4

return
<rooms>
{
        for $r in (1 to $room_count)
                let $dist := <distribution><symbolic>
                                <gaussian mean="{$r}" variance="{$variance}"/>
                             </symbolic></distribution>
                let $dist_hist := symbolic:createHistogramFromSymbolic($dist, $delta)
                return
                <room>
                        <number>{$r}</number>
                        <temperature>{$dist_hist}</temperature>
                </room>
}
</rooms>
```

Figure 6.2: *XQuery code used for generating a test document.*

```
let $roomsdoc := doc("/home/misc_cis/scholte/msc_project/cul/version2/output.xml")/rooms

return
<rooms>
{
        for $room in $roomsdoc/room
                return
                <room>
                        <number>{data($room/number)}</number>
                        <mean_temp>
                          {prob:v_MEAN(exactly-one($room/temperature/distribution))}
                        </mean_temp>
                </room>
}
</rooms>
```

Figure 6.3: *XQuery code used for measuring the performance of querying an XML document containing continuous uncertain data.*

discussed before, the interval is defined as [(mean - 3 * standard deviation), (mean + 3 * standard deviation)]. When the variance equals 4, the interval is $[-6, 6]$, the length is 12. In the second test, the variance is the variable. The variance changes with a step size of 5, starting at 5 ending at 50. The second test is carried out twice: once with a delta of 0.05 and once with a delta of 0.1. As with the first test, the second will be run three times, so that execution times can be averaged. The benchmark has been implemented using the XQuery language. The code is printed in figure 6.1. In section 5.5 we described a mechanism that prevents

the Pr operator from executing when the interval, which is specified by the user, cannot be covered completely by histogram segments. This mechanism will be disabled during testing. However, we will determine the number of histogram segments that are processed by each query.

Another benchmark concerns the performance assessment of querying an XML document containing numerous continuous distributions. The query execution time of querying a different number of distinct continuous distributions, is measured. The benchmark consists of two tests. Each test computes the expected value of each Gaussian distribution captured by the XML document. In the first test, the delta is equal to 0.01 where in the second test the delta equals 0.1. In all tests, the variance of the Gaussian distribution remain the same, namely 4. In each test, the expected value from a increasing number of continuous distributions is computed: 1, 10, 20...100 distributions. The XML document containing a prespecified number of Gaussian distributions, is automatically generated using an XQuery script. The code of this script is shown in figure 6.2. The script takes care of generating a number of different Gaussian distributions with corresponding histograms. The efficiency of executing the mean operator on Gaussian distributions in an XML document, will be measured by another XQuery script. The code of this script is depicted in figure 6.3.

6.1.2 Aggregate operators

The complexity of all the aggregate operators is the same. Therefore, it is not necessary to assess the efficiency of all the aggregate query operators. The most expensive operation is the cartesian product over the discretized continuous distributions. The benchmark will simulate scenarios in where 2 or more different Gaussian distributions are aggregated using the ASUM operator. The performance is measured by the query execution time. The Gaussian distributions, used in each test, will differ with respect to the mean, the variance remains the same (and thus the interval lenght of the histogram will be the same in all tests). To summarize: this benchmark constists of 4 tests, the number of distributions varies between 2 and 5 and the delta of the histograms will be equal to 1.0.

6.1.3 Test environment

Each individual test was carried out by executing an XQuery script on a MonetDB/XQuery installation. Detailed information about the installation can be found below. The script was executed using the following command: `mclient -lx script.xq --time`. The `--time` parameter will produce a result like this:

```
<example>Hello World</example>

Trans      18.000 msec
Shred       0.000 msec
Query       5.000 msec
Print       0.000 msec
Timer      24.436 msec
```

The Query time is taken as a quantity for the performance (efficiency).

The following hardware was used to conduct the experiments:

Table 6.1: The results of the benchmark that simulates the use of value-based operators with the delta being the variable, parameters used: mean=0.0, variance=4.0.

Delta	Segments	Error	Average query time (ms)
0.005	1200	0.00135267471431571	10356.450
0.010	600	0.00135546539379205	1706.153
0.015	400	0.00135826992257792	735.304
0.020	300	0.0013610883004152	481.143
0.025	240	0.00136392052693817	345.081
0.030	200	0.00136676660168072	273.499
0.035	171.4285714	0.001403689318692	225.954
0.040	150	0.00137250029343422	200.604
0.045	133.3333333	0.001409581072012	178.765
0.050	120	0.00137828936985912	166.073
0.055	109.0909091	0.00139256081887861	156.398
0.060	100	0.00138413382347729	148.148
0.065	92.30769231	0.00143319235132027	142.916
0.070	85.71428571	0.0015081785933817	137.946
0.075	80	0.0013930043156935	134.038
0.080	75	0.00139598882407138	131.178
0.085	70.58823529	0.0015178206631406	122.966
0.090	66.66666667	0.00154595651280353	119.162
0.095	63.15789474	0.00143988711926584	118.912
0.100	60	0.00140806520216291	118.060

- Processor: AMD Opteron(tm) Processor 246
- 8 Gigabytes of main memory

The following software was used:

- Linux version 2.6.22.5-31-default (geeko@buildhost) (gcc version 4.2.1 (SUSE Linux))
- MonetDB Server v4.22.0
- GDK v1.22.0
- MonetDB/XQuery module v0.22.0

6.2 Results

The first experiment measures the accuracy and the efficiency with the delta and variance being the test parameters. In order to determine a valid query execution time, all queries in all tests in this experiment have been executed 3 times and the average has been taken from the query execution times. The query execution times that are indicated in the tables have been measured in milliseconds. The results from the first experiment are shown in table 6.1 and table 6.2. The first table corresponds with the test where the segment size is the variable, the variance is fixed. Table 6.2 shows the result of the tests in which the variance is the variable. Table 6.2(a) shows the result from the test in which a segment size of 0.05 has been used, the results showed by table 6.2(b) have been produced with a segment size of 0.1.

Table 6.2: The results of the benchmark that simulates the use of value-based operators with the variance being the variable.

(a) Parameters: mean=0, delta=0.05

Variance	Segments	Error	Average query time (ms)
5.00	134.1640786	0.00139185592373008	175.392
10.00	189.7366596	0.00142085546513198	247.074
15.00	232.3790008	0.00138644436868173	324.034
20.00	268.3281573	0.00137891197585915	387.629
25.00	300	0.0013610883004152	477.604
30.00	328.6335345	0.00138613718956187	539.303
35.00	354.964787	0.00139615654017777	604.008
40.00	379.4733192	0.0013755028563922	671.112
45.00	402.4922359	0.0013746594774226	746.740
50.00	424.2640687	0.00136613060561341	835.287

(b) Parameters: mean=0, delta=0.1

Variance	Segments	Error	Average query time (ms)
5.00	67.08203932	0.00141858258894539	119.415
10.00	94.86832981	0.00151595111459518	141.343
15.00	116.1895004	0.00140152063449117	159.175
20.00	134.1640786	0.00139185592373008	177.308
25.00	150	0.00137250029343427	198.883
30.00	164.3167673	0.00139670957417792	216.578
35.00	177.4823935	0.00140599471663555	232.219
40.00	189.7366596	0.00142085546513198	247.932
45.00	201.246118	0.00138319555101257	266.464
50.00	212.1320344	0.00137417609631973	292.977

The second experiment involves measuring the performance of querying an XML document containing an increasing number of distinct Gaussian distributions. In this experiment, the mean operator is executed on all the Gaussian distributions returning a list of expected values. The XML document contains the symbolic as well as the histogram representation of each Gaussian distribution. The variance of each Gaussian distribution in the XML document is the same: 4.0, the mean differs. The results of the experiment are shown in table 6.3. The first table shows the result of the experiment with delta=0.01 being the segment size of the histogram. It shows the number of Gaussian distribution with the corresponding average query execution time in seconds. The average query execution times have been computed by taking the average over the query execution times from 3 tests. The second table shows the result of a similar experiment, the value of the delta in this experiment was equal to 0.1. The query execution times indicated in the second table have been measured in milliseconds.

The goal of the third experiment is to determine the relationship between performance and the number of different continuous distributions that are aggregated by a query containing an ASUM operator. The continuous distributions are Gaussian distributions having a variance

Table 6.3: The results of the benchmark that simulates the querying of a document.

(a) Parameters: delta=0.01

Distributions	Average query time (s)
1	5.138
10	56.120
20	152.234
30	223.966
40	397.821
50	491.174
60	586.109
70	900.544
80	1020.830
90	1031.545
100	1160.790

(b) Parameters: delta=0.1

Distributions	Average query time (ms)
1	84.388
10	298.213
20	576.600
30	943.741
40	999.743
50	1383.812
60	1593.433
70	1729.483
80	2021.038
90	2375.604
100	3883.425

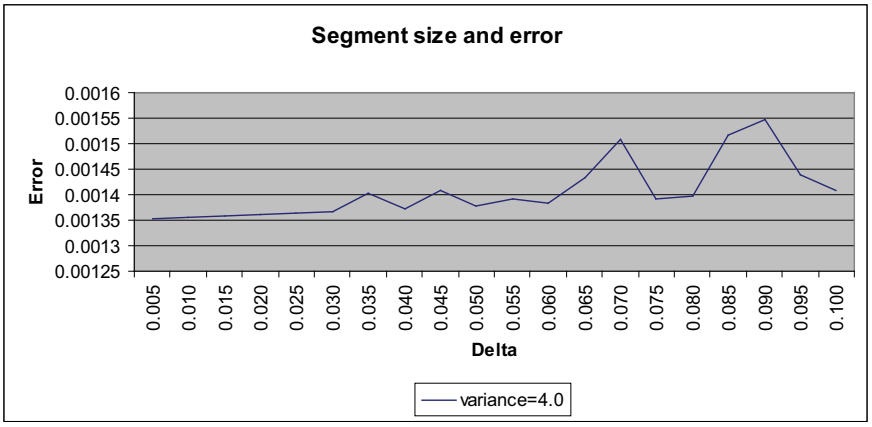
Table 6.4: The results of the benchmark that simulates the use of aggregate operators. The segment size was equal to 1.0

Distributions	Average query time (s)
2	0.338
3	4.487
4	179.417
5	23541.323

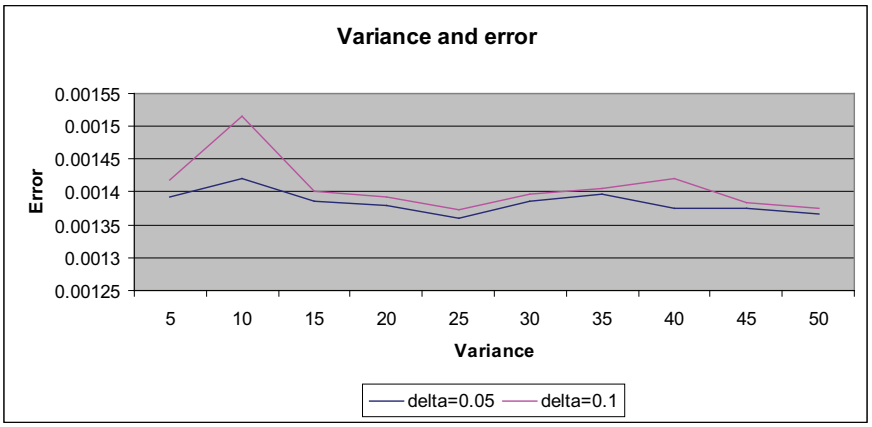
of 4. The mean of each Gaussian distribution differs, distribution number one has a mean of 0, distribution number two a mean of 1, distribution number three a mean of 2 and so and so on. Each test is executed 3 times, the indicated query execution time is the average that has been taken over the individual query execution times.

6.3 Analysis

The first experiment was conducted in order to give an answer to the question of how the variance-parameter of a Gaussian distribution and the delta-parameter of a corresponding histogram affects the accuracy and performance of query operators, especially the Pr operator. The tables containing the results of the first experiment were transformed to graphs for analysis purposes. The pairs in the tables correspond to points in the graphs, the points are connected by lines resulting in graphs. The graphs in figure 6.4 show the impact of an increasing segment size on the accuracy of the query answer and the impact of an increasing variance on the accuracy, respectively. The graphs shown by figure 6.6 show the relationship between the segment size and the response time and the relationship between variance and response time, respectively. The impact of an increasing variance on the accuracy and the



(a) Segment size and error, variance=4.0



(b) Variance and error, delta=0.05,0.1

Figure 6.4: *Query accuracy of computing a probability.*

response time, was measured with two different values for the segment size.

The graph in figure 6.4(a) has been derived from table 6.1 and depicts some local maximum values. The graph shows the relationship between the segment size and the deviation between the query answer and the exact answer when computing the probability $P(0 \leq X \leq \infty)$. Since the expected value is equal to 0, the exact probability should be equal to 0.5. When querying the histogram representation, querying for $P(0 \leq X \leq \infty)$ means that the area under the probability density function is computed corresponding to the following interval: $[0, 6]$; the variance is 4 and the right border is equal to the expected value plus three times the standard deviation. The local maxima are caused by the fact that for some values for the delta, there is no round number of segments that lie on the interval [0, 6]. This is also shown by the second column of table 6.1, the values in the field "Segments" have been computed by dividing the length of the interval by the segments size. When there is no round number of segments that cover the interval, there is a significant larger error in the answer of a probabilistic query. This is because some (additional) surface below the graph of a probability density function is in the computation omitted, the number of segments processed by the algorithm is floored.

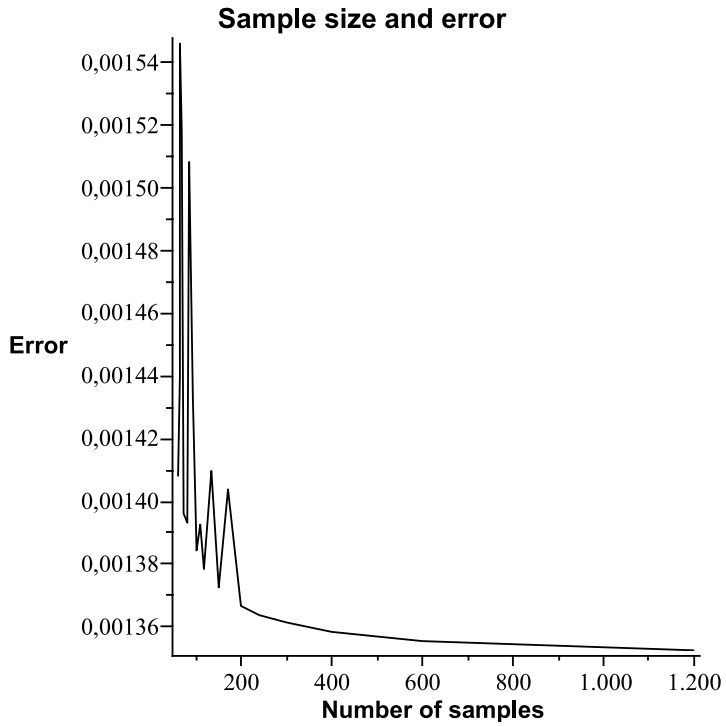
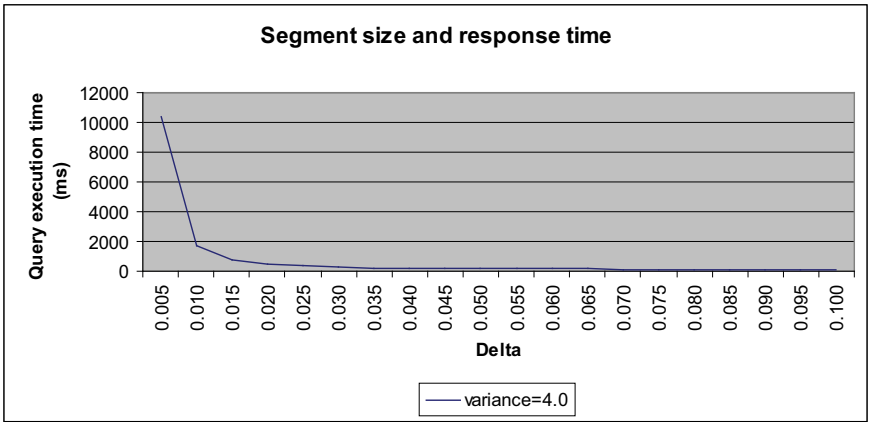


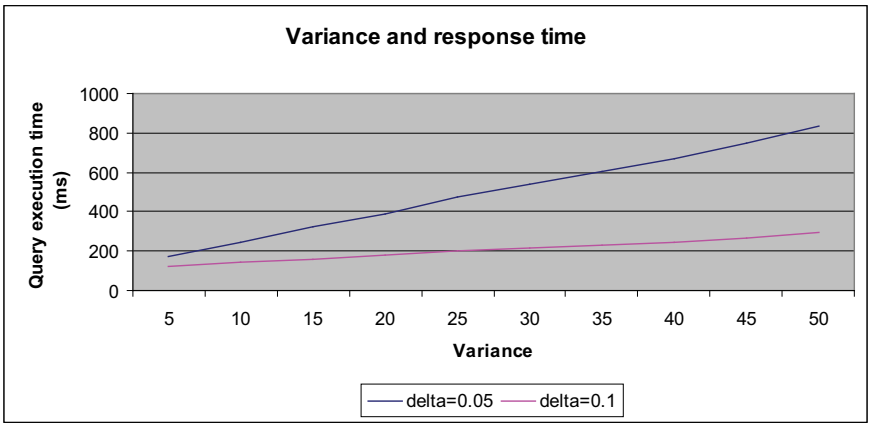
Figure 6.5: *The number of samples (segments) and the error.*

The algorithm prevents that one segment crosses one of the borders of the interval during computation of an answer. If we omit the local maxima in the intrepretation of the graphs, then we can conclude from the graph in figure 6.4(a) that the error increases linearly with the segment size. As aspected, we see in figure 6.4(b) that a smaller segment size yields a smaller error. Furthermore, we see that an increasing variance does not have a major impact on the accuracy of the query answer. This is sound since there is a constant level of error which caused by the cut-offs of the probability density function on the right-hand and left-hand side. Another source of inaccuracy are the casting and rounding errors and the imprecision caused by the underlying XML database management system. An example is the assignment of the value 0.0000001 to a variable and returning that variable gives the following result: 0.000000. To summarize, inaccuracy is caused by:

- Histogram segments that do not cover an interval completely. The number of histogram segments that are taken into account is less than the size of the given interval.
- The interval of a continuous distribution is not closed where the interval of a histogram is closed. In case of a Gaussian distribution, 0.27 percent of the probability mass is missing.
- Histogram segments that cross the graph of the probability density function (Gaussian).
- Rounding casting errors made by the underlying XML database management system. Furthermore, the DBMS does not support large fractions. When using the double data type, a fraction larger than 6 digits is not supported by the DBMS.



(a) Segment size and efficiency, variance=4.0



(b) Variance and efficiency, delta=0.05,0.1

Figure 6.6: *Efficiency of computing a probability.*

Figure 6.5 shows the relation between the number of segments and the error. The graph has been derived from table 6.1. The error decreases exponentially when the number of segments increases. This behavior is in accordance with the evaluation results of the prototype developed in the Orion project [22].

Another goal of this experiment was to examine the relation between the segment size of a histogram and the response time on one hand and on the other hand the influence of an increasing variance on the response time. Figure 6.6(a) shows the effect of an increasing segment size on the performance. It is clear that the query execution time decreases exponentially when the segment size increases. In other words: the performance increases exponentially when the segment size increases. Figure 6.7 shows the relation between the number of sample points (number of segments) and the query execution time. The figure is derived from table 6.1. The relation can be described by an exponential function.

The graph in figure 6.6(b) depicts the relation between variance and performance, each line corresponds with a different segment size. We see that when the variance grows, the query execution time increases proportionally. As aspected, histograms with a larger segment

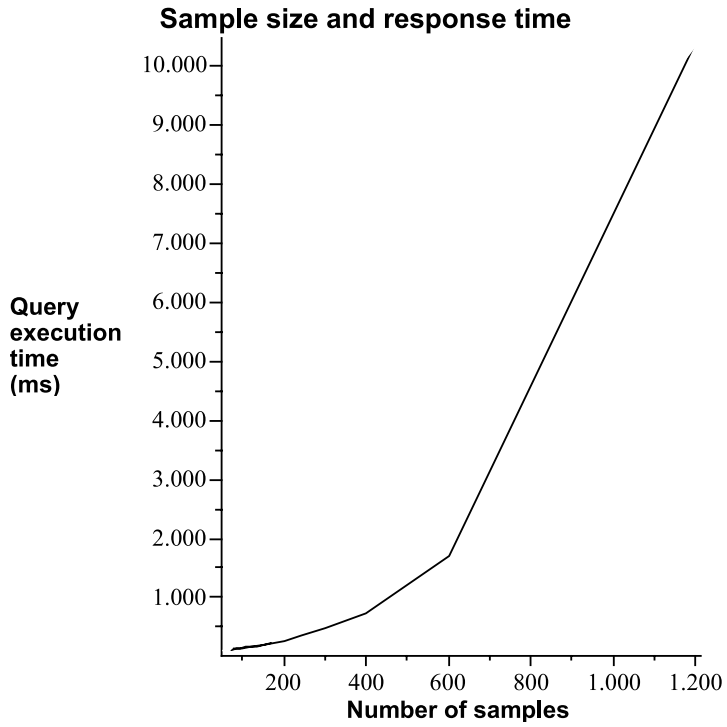
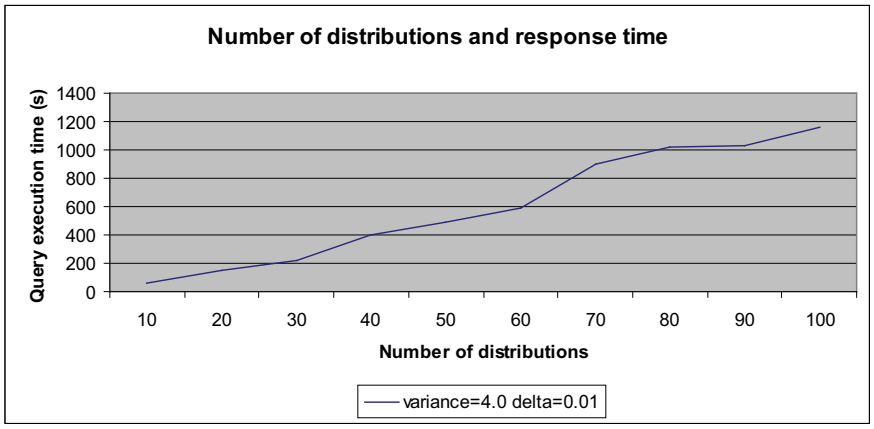


Figure 6.7: *The number of samples (segments) and the runtime.*

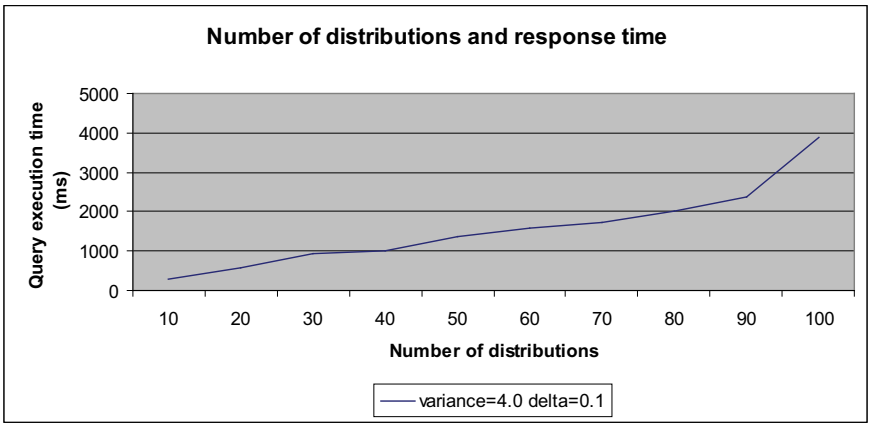
size outperforms the histograms with smaller segment sizes.

The goal of the second experiment was to examine the relation between the query performance of querying a document containing numeruous disctinct continuous distributions. Figure 6.8(a) shows a graph of table 6.3(a) that was generated in a similar way as in the previous experiment. Figure 6.8(b) shows a similar graph, it has been derived from table 6.3(b) and the delta is larger: 0.1. We can conclude from the graphs that the query time grows linearly with the number of Gaussian distributions that are processed when querying a document containing those distributions. The behavior is in accordance with the evaluation results of the prototype developed in the Orion project [22]. Furthermore, we see a big difference in query execution time between the different segment sizes. When querying histograms with the smaller segment size, the query execution time is much greater than querying the histograms with the bigger segment size.

Table 6.4 and figure 6.9 show the results of the benchmark that measures the performance of aggregate queries. The number of continous distribution that are aggregated is the variable. We see that the execution time increases a lot when 5 distributions are aggregated. We performed the tests with a relative large delta, 1.0. Due to a lack of resources in terms of available hardware and the amount of time available, it was not possible to perform tests that aggregate more than 5 distributions. For the same reason, it was also not possible to perform tests using a smaller delta. This prevents us from making satisfying conclusions about the measurements.



(a) Document size and query execution time, delta=0.01



(b) Document size and query execution time, delta=0.1

Figure 6.8: *The relation between the number of Gaussian distributions and query execution time.*

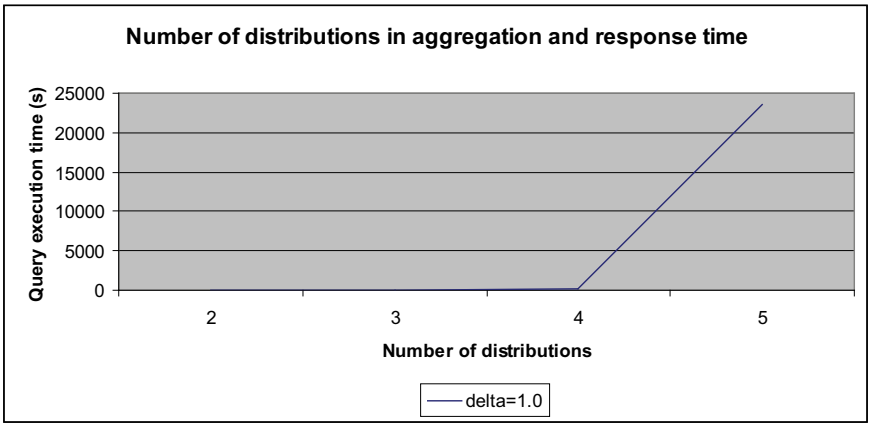


Figure 6.9: *The relation between the number of Gaussian distributions in an aggregate query and query execution time.*

Chapter 7

Conclusions

7.1 Summary

In this thesis, we introduced a probabilistic data model that supports the representation of discrete and continuous uncertain data in XML. Continuous probability theory was used as a reference for the development of this probabilistic data model. The data model supports the representation of standard as well as non-standard continuous distributions. We (re-) defined the semantics of queries on continuous uncertain data in terms of possible worlds, based on the work presented in [12, 23], and we showed how the semantics are followed when evaluating queries on Probabilistic XML containing continuous uncertain data. Next, we introduced some new operators for aggregating continuous distributions based on possible world semantics. The mechanism we presented, allows the selection of a possibility from each combination of possibilities which originates from continuous distributions and merge the possibilities that have the same values. We also introduced an approach for measuring the efficiency and accuracy of a DBMSs that has support for the management of continuous uncertain data.

7.2 Research questions

In this section, we answer the research questions posed in section 1.3.

1. *Which additions to existing data models are necessary in order to support the representation of continuous uncertain data in XML?*
   In chapter 3, we presented a data model for probabilistic XML that supports the representation of continuous uncertain data using the continuous representation. The continuous representation is an extension to the compact representation proposed in [12, 23]. We showed that the continuous representation and the compact representation are equivalent since the continuous representation can be transformed to the compact representation. As a result of this, all the properties that are valid for the compact representation, are also valid for the continuous representation. Continuous probability distributions are stored by the continuous representation as the name of one of the corresponding possible probability density functions and its parameters. The probabilistic model supports the storage of beta, gamma, gaussian and uniform distributions. We consider the support for the representation of joint and conditional

probability distributions, dependencies between probabilistic attributes and lineage in the probabilistic data model, as future work.

2. *Which semantic foundation is suitable for storing continuous uncertain data?*
   We have explained the possible world approach in chapter 2 and explained how, following this approach, discrete data can be stored using the probabilistic XML data model. With this foundation laid, we introduced how the semantics are followed when storing continuous uncertain data. A continuous uncertain data item is described by a probability density function and it reflects an infinite number of possibilities and thus also possible worlds. We showed that a probabilistic XML document containing probability density functions (continuous representation) can be transformed to the compact representation. Then, each probability node in the compact representation has an infinite number of possibility nodes as children.

3. *How can we provide support for querying continuous uncertain data?*
   We introduced some simple query operators (chapter 4) for computing probabilities, expected values and variances. The operators can be used in XQuery queries to query probabilistic XML data. We showed that the operators can be applied in predicate queries.

   How should the semantic foundation be applied in order to support querying continuous uncertain data in an intuitive way?
   In chapter 3 we showed that it is theoretically possible to transform the continuous representation to the possible world as well as the compact representation proposed in [12, 23]. Since these representations can be queried following the possible world approach, as shown in [12, 23], we are able to show in chapter 4 that continuous uncertain data can be queried following the possible world approach.

   Which query operations are useful when querying continuous uncertain data?
   In order to allow the user to integrate or aggregate two or more continuous distributions into one single continuous distribution, aggregate operators were introduced. The aggregate operators are based on possible world semantics. We can distinguish three phases in the process of aggregating distributions: the cartesian product, the selection phase and the merging phase. The application of the cartesian product yields an enumeration of combinations of possibilities, an enumeration of possible worlds; each possibility originating from a distribution is combined with all the possibilities from all the other distributions. The type of selection operator used in the selection phase determines the name of the aggregate operator. The selection phase selects one value from each combination of possibilities. At the same time, the probabilities, corresponding to the possibilities in each combination, are multiplied. Thus, in the selection phase, one possibility is selected from each possible world. The last step concerns the merging of value-probability pairs that have equal values.

4. *How can the theoretical concepts as a result from the research on the management of continuous uncertain data be practically applied?*
   We described in chapter 5 the development of a proof of concept. The proof of concept supports the storage, querying and manipulation of Gaussian distributions. It was possible to implement the proof of concept using the XQuery language. We introduced a new representation, the argumentation can be found in chapter 5. This representation

stores the symbolic form of a probability density function along with the corresponding histogram. The implementation of all the query operators work directly on the histogram representation of continuous uncertain data. We also demonstrated the use of the prototype.

5. *What is the behavior of a probabilistic XML DBMS supporting the management of continuous uncertain data in terms of efficiency and accuracy of query answers?*
   We developed three benchmarks that are able to assess the query execution times and the inaccuracy of query answers. The first benchmark compares the answer of a probability query operator with the exact probability. Both probabilities should be equal since they correspond to the same interval of the same continuous distribution. This benchmark measures at the same time the performance. The error grows proportionally with the segment size. Sources of inaccuracy are:
   - Histogram segments that do not cover an interval completely. The number of histogram segments that are taken into account is less than the size of the given interval.
   - The interval of a continuous distribution is not closed where the interval of a histogram is closed.
   - Histogram segments that cross the graph of the probability density function (Gaussian).
   - Rounding casting errors made by the underlying XML database management system. Furthermore, the DBMS does not support large fractions.

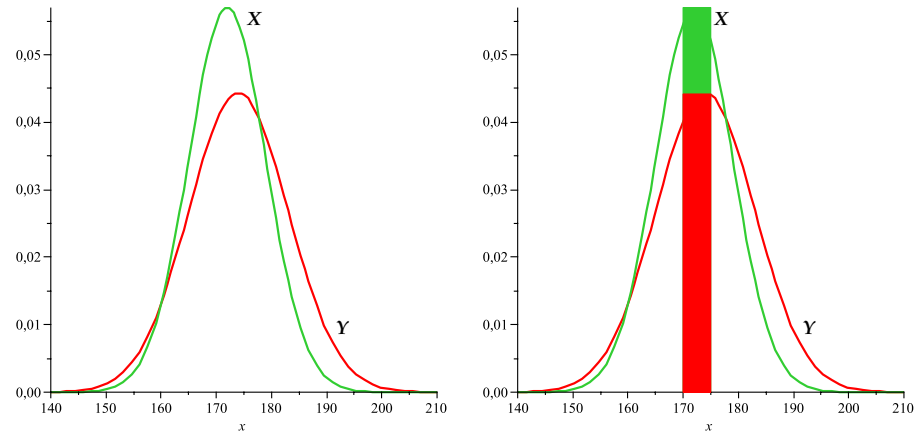
   The performance increases exponentially when the histogram segment size increases, the performance increases proportionally with the variance. The second benchmark assess the performance of querying a probabilistic XML document of different sizes containing different Gaussian distributions. We have found that the response time increases proportionally with the number of distributions in the probabilistic XML document. The third benchmark assesses the performance of the aggregate operators by applying an aggregate operator over a growing number of different Gaussian distributions. This showed us that aggregate operators are really computational expensive operations. The method of evaluating did not satisfy us as the benchmarks do not reflect a real-life scenario. For this reason, we will address in section 7.3 the need for benchmarks that can be used to benchmark relational as well as semi-structured probabilistic database management systems, capable of managing continuous uncertain data.

7.3 Future work

This thesis would not be complete without a section on future research directions. In this section, we try to explain problems which have been become visible during our research. Furthermore, we will motivate the need for examining these issues and we will give some directions.

7.3.1 Support for querying multiple stochastic variables

At this moment, the query operators in the continuous uncertain language supports only querying one attribute that has been modeled as a continuous distribution. However, when



(a) Two probability density functions of X and Y

(b) A segment from the unification of X and Y

Figure 7.1: *Unifying two continuous distributions.*

computing a probability, it is common practice that multiple stochastic variables are involved. Typical examples include the computations of synchronous probabilities and conditional probabilities. In this section, we will give some examples of queries where multiple stochastic variables are involved. Moreover, we will go into detail about the consequences for the query language and the data model.

Unifying continuous distributions

Consider figure 7.1(a). The figure depicts two Gaussian distributions that correspond with the stochastic variables X en Y. X and Y represent both the body lengths of males in one country. The Gaussian distribution of variable X has parameters mean 172 and variance 49, the distribution of Y does have the parameters mean 174 and variance 81. The Gaussian distribution corresponding to stochastic variable X has been constructed by measuring the body length of hundred people, where Y has been constructed by taking a different population, having a size of one million people. Because the distribution corresponding to variable Y is based on more experiments, it's model is more accurate than the distribution of X. However, when the two distributions will be taken together, even more experiments are involved which will improve the accurateness.

An example of the unification of two distributions is shown in figure 7.1(b). Let's denote the resulting continuous distribution with the variable L. The area of the segment is an approximation of the probability $P(170 \leq L \leq 175)$. This probability can be computed by the following formula: $P(170 \leq L \leq 175) = P(170 \leq L \leq 175|group1) * P(group1) + P(170 \leq L \leq 175|group2) * P(group2)$. When we consider that group1 is represented by stochastic variable X and group2 by stochastic variable Y, the probabilities $P(170 \leq L \leq 175|group1)$ and $P(170 \leq L \leq 175|group2)$ correspond to the area of respectively the green and red rectangles. The probabilities $P(group1)$ and $P(group2)$ are the weighing factors in the resulting distribution. The weighing factors can be determined by the size of the population. In this example $P(group1)$ is equal to $\frac{1000}{(1000+1000000)}$ and $p(group2) = \frac{1000000}{(1000+1000000)}$. The size of populations or

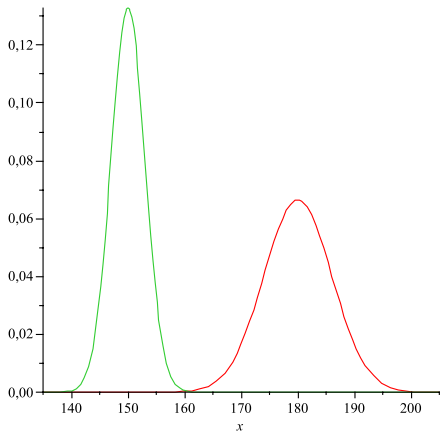


Figure 7.2: *Almost no overlap.*

experiments are not *part* of continuous distributions, though, the continuous distributions are *created* using the population. This is where lineage comes into play. To determine the weighing factors, we need to where the continuous distributions are originating from. We go into detail on lineage in one of the succeeding sections.

The open issues related to unification are:

1. Query language
   The query language could be extended with a UNION operator. This UNION operator can be used by the end-user to unify two continuous distributions, posing queries on the result, present the result on a screen or to insert the result in the database. The UNION operator takes the location of two continuous distributions as argument. An alternative approach is to allow the end-user to use a UNION construct directly in the Pr operator. Then, the end-user has to specify in the Pr operator the location of the stochastic variables to query. With this approach, the union of two continuous distributions is computed at query time. Whether this is a feasible approach is currently not known.

2. Data model and representation (histogram)
   The unification of two continuous distributions yields a non-standard continuous distribution. The step size is known a-priori. So the result of an unification can be represented by a histogram. Suppose that there is a large gap between the two continuous distributions, i.e. there is not much overlap (as depicted in figure 7.2). How to represent this is by a histogram? One approach is to floor the original distribution on the interval where the density of the probability mass is small. Then, the interval between the two distributions can be represented in the histogram by zeros for the y-values on that interval. Following this approach, and there is a huge gap between the distributions, the resulting histogram can become very large.

Joint probability distributions

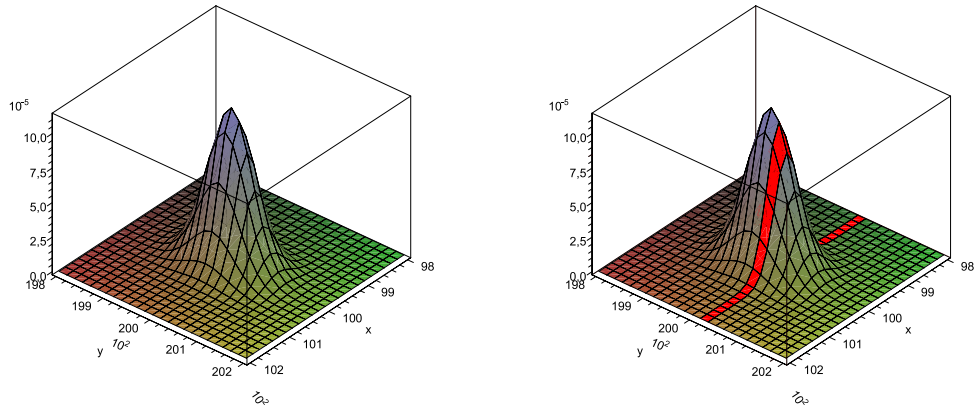
Consider the scenario of the beer brewery given in section 4.3.1. The production facility does have two production lines. The daily capacity of each production line is expressed by the amount of bottles filled and it can be modeled by a Gaussian stochastic variable. For the capacity of production line one, holds the following parameters mean=10000 and variance=1000, the capacity of production line two can be described by a Gaussian distribution with parameters mean=20000 and the variance=2000. Suppose we would like to know the probability that, on a random day, production line one produces more than 11000 bottles and production line 2 produces more than 22000 bottles. If the amount of bottles filled in production line one is described by stochastic variable X and the amount of bottles filled in production line two is described by stochastic variable Y, the probability can be expressed as follows: $P(X > 11000 \; and \; Y > 22000)$. These kinds of probabilities are called joint probabilities. When the distributions of X and Y are taken together, we get the joint distribution of X and Y with an associated joint probability density function: $\int_x \int_y f_{X,Y}(x, y) dy \, dx = 1$.

A common example of a joint probability density function is the multivariate normal distribution, it is a generalization of the one-dimensional Gaussian distribution to multiple dimensions. A special case is the bivariate normal distribution for which the filling-capacity of two production lines is a good example. The bivariate normal distribution in the example above, is drawn in figure 7.3(a).

Figure 7.3(b) depicts a bivariate normal distribution. The red rectangle cuts the bivariate normal distribution on the y-axis. The volume under the red rectangle corresponds to the probability $P(9800 < X < 10200 \; and \; 20020 < Y < 20040)$. When the width of the red rectangle becomes infinite large, the volume below the rectangle corresponds to the probability $P(9800 < X < 10200 \; and \; Y = j) = P(9800 < X < 10200)$. This is an example of a marginal probability. The probability can be computed using the marginal probability density function. This function can be determined by integrating (or summing in the discrete case) over the variable Y: $f_X(x) = \int_y f_{X,Y}(x, y) dy$. This is a basic operation and can be supported by the DBMS by implementing a marginalize function. Given a probability density function over a set of attributes, the marginalize function returns a new probability density function over a subset of attributes.

We came up with an example where the stochastic variables are independent. However, in many situations the stochastic variables are dependent and they may be correlated too. An open issue is how to model these statistical dependencies in semi-structured data and how to encode correlations and factors in this data.

Because there are many scenarios in which end-users would like to use joint probabilities, we would like to have support for this feature in our probabilistic XML database management system. The theoretical data model presented in chapter 3 can be extended in such way that it will have support for joint probability density functions. The data representation used by the prototype should be adapted as well. The histogram representation will not be sufficient anymore because it only supports the representation of one-dimensional data where the storage of multi-dimensional data will be required then. As stated above, a complicated aspect will be the modeling of statistical dependencies and correlations in a semi-structured probabilistic data model. Having said all this, it is actually necessary to investigate the need for storing multi-dimensional probability density functions in semi-structured data. An alternative (or optional?) approach would be to store the individual stochastic variables with its probability density function and the correlation factors and to



(a) A bivariate normal distribution.

(b) A sliced bivariate normal distribution.

Figure 7.3: *Bivariate normal distributions.*

compute the joint probabilities at query-time. Besides the data model and the representation, another open issue involves the query language. The query language requires additional constructs in order to support querying joint probability density functions. It should have at least support the "AND" construct in the Pr operator such that Pr($X > 2$ *and* $Y > 3$) constructs are allowed. When the probabilistic density functions of two stochastic variables, X and Y, are stored separately in an XML document (as in the alternative approach), the location of X and Y should be specified within the query. Instead of just providing the attribute-names, the location can be provided using an XPath expression. An example could be Pr(`room[1]/temperature>18 and /room[2]/temperature>19`) while the XML document contains temperatures of all rooms in an office building. Moreover, needless to say, when the DBMS supports the storage of multi-dimensional probability density functions, the query language should have support for marginalization in order to rule out stochastic variables.

Multiple (exclusive) events

Consider again the scenario of the beer brewery given in section 4.3.1. Suppose someone would like to know the probability that in production line one more than 22000 bottles are filled or that in production line two more than 11000 are filled. This is an example of two mutual exclusive events that either occur. The probability can be computed as follows $P(X \geq x \text{ or } Y \geq y) = P(X \geq x \bigcup Y \geq y) = P(X \geq x) + P(Y \geq y)$. In case the two events are not mutual exclusive, the probability can be computed as follows $P(X \geq x \text{ or } Y \geq y) = P(X \geq x) + P(Y \geq y) - P(X \geq x \text{ and } Y \geq y)$.

One way to support this, is to extend the query language with an "OR" construct that can be used within the argument which is provided to the Pr operator. Furthermore, the query language should have some kind of rewrite mechanism such that it will support the transformation described above. As discussed in the previous section, the query evaluation component of the DBMS should have means to locate the stochastic variables X and Y in the XML document. The stochastic variables could reside in entirely different sections of the XML document. One approach to solve this problem is to express the location of the variables X and

Y as XPath expressions that will be evaluated individually by the Pr operator. Another aspect concerning the probability computation of two mutual exclusive stochastic variables has to do with checking whether those two probabilistic attributes are indeed mutual exclusive. The data model of probabilistic XML handles mutual exclusiveness in a natural way. Two events are mutual exclusive if the associated stochastic variables reside in two different poss-nodes. One approach to solve this issue, is to determine the relative locations of the stochastic variables.

Conditional probabilities and distributions

A conditional probability is the probability of event A occurring given the occurrence of an event Y. According to Bayes theorem, the following holds: $P(X \mid Y) = \frac{P(X \text{ and } Y)}{P(Y)} = \frac{P(Y \mid X)P(X)}{P(Y)}$. Consider the scenario of the beer brewery. Suppose someone would like to know the probability that the throughput of production line one is more than 10000 bottles a day while the number of broken bottles caused by the caps-pushing machine is 50. This can be expressed with the following conditional probability: $P(X \geq 10000 \mid Y = 50)$. X and Y are both Gaussian distributed and it is likely that the variables X and Y are dependent and negatively correlated. Let's say the correlation between X and Y is -0.01. We can compute the conditional distribution using linear regression. We use the following formulas:

- $\mu_{X|Y=a} = ba + c$
  $b = \frac{\rho_{XY}\sigma_X}{\sigma Y} = -0.01$
  $c = \mu_X - b\mu_Y = 10000 - (-0.01 * 50) = 10000.5$
- $\sigma^2_{X|Y=a} = (1 - \rho^2_{XY})\rho^2_Y$

The resulting conditional (Gaussian) distribution will have the parameters $\mu = -0.01 * 50 + 10000.5 = 10000$ and $\sigma^2 = 1 - \rho^2_{XY}\sigma_X = 1 - (0.0001 * 1000) = 0.9$. This distribution corresponds to a "cut through" of the multivariate Gaussian distribution on the y-axis at position $y = 50$. An example of a multivariate Gaussian distribution can be found in figure 7.3. The conditional probability density function can be written as: $f_{X|Y}(x|y) = \frac{f_{X,Y}(x,y)}{f_Y(y)}$. Note that if $f_{X|Y}(x|y) = f_X(x)$, X and Y are said to be independent.

Support for conditional probabilities and distributions by a probabilistic DBMS is desirable because there are many foreseeable scenarios in which people deal with conditional probabilities. Supporting them has the following implications for the DBMS:

1. Data model
   As [19] and [23] discovered, dependencies between probabilistic attributes, like mutual exclusiveness and simultaneous occurence, can be expressed in XML in a naturual way. However, when probabilistic attributes are dependent, they might be correlated as well. The correlation coefficient indicates to what extent two random variables are correlated. The data model we developed does not have support for modeling correlations and their coefficients.

2. Query language
   It would be interesting to extend the query language with some construct that can produce a conditional probability density function given two probabilistic attributes which are described by continuous distributions. The result should be a symbolic

representation with a corresponding approximation (e.g. histogram). Besides that, it is desirable that the Pr operator in the query language will be adapted in such way that the usage of "|" constructs are allowed. Another related issue is how correlations between probabilistic attributes fit into possible worlds semantics. This would also be an interesting topic.

The data model and query algebra proposed in [26] supports the storage, manipulation and querying of joint and conditional probability distributions in semi-structured data. However, the model and query algebra is limited to discrete data. The results of this project might be used to extend our data model and query language for continuous uncertain data.

7.3.2 Support for lineage

Lineage refers to the name of a mechanism in a DBMS that *keeps track of where the data came from*. Our prototype supports one kind of lineage. The prototype is able to keep track of operations performed on each base probability density function corresponding to each stochastic variable (probabilistic attribute). However, lineage is a much broader topic. It involves interrelating representations of different real world objects by links, each having its own semantical meaning. Benjelloun et al. distinguishes in [4] internal and external lineage. As mentioned before, internal lineage refers to references between representations of real world objects in one database where external lineage refers to references to representation that are outside the database. Support for lineage (internal and external) is desirable as the example in section 7.3.1 made clear. Moreover, lineage allows us to track and recover the original data when performing probabilistic computations; this may improve the quality of the query result in some situations.

As [9] shows, it is possible to encode *data provenance* or *lineage* into XML documents. The open issues concern the modeling of lineage with continuous uncertainty as an alternative to the confidence values presented in [4].

7.3.3 Query optimization

Until now we have not much worried about query performance, including speed and accuracy, of the probabilistic XML DBMS. As shown in chapter 6, there is a trade-off between accuracy and response time. The goal of query optimization is to improve response times of the DBMS as well as accuracy of the query answers returned by the DBMS.

A relational query can be optimized using a query optimizer. The two main components of a query optimizer are the query execution plan generator and the plan cost estimator. A query plan defines the way of how a query is evaluated. In a relational DBMS, the query plan can be seen as a relation expression with evaluation methods attached to each operation. The query plan generator generates a number of query execution plans using commutativity and associativity rules. A query optimizer uses a number of heuristics and cost estimates in order to choose among query execution plans. The query is evaluated according to the query execution plan chosen. It is likely that the execution of probabilistic queries can be optimized using a query optimizer. It might be that by developing commutativity and associativity rules for probabilistic operators, probabilistic queries can be optimized. Another issue to be examined is how we can estimate the costs of executing a query execution plan.

Another aspect of query optimization is the following. As described in chapter 5 the implementation of all the query operators process continuous uncertain data that has been discretized. This approach ensures that all kinds of continuous distributions (also non-standard continuous distributions) can be queried following possible world semantics. However, it is not an very efficient approach because the query operators have to process a large number of histogram segments during query execution. In many situations, the parameters from the symbolic representation can be used in conjunction with some sophisticated functions to compute a query answer. For instance, the cumulative distribution functions of continuous distributions can be used for computing probabilities. In many cases, the expected value and the variance can be computed, using a distribution-specific function with parameters which are encoded in the symbolic representation. A good example is the computation of the expected value from a continuous uniform distribution; it is simply $\frac{a+b}{2}$ where parameters a and b will be stored in the symbolic representation. The function that computes the expected value is distribution-specific and the function has to be present in the DBMS. The functionality can be implemented by a mean-function in the DBMS accepting the symbolic representation of the continuous uniform distribution, containing the parameters a and b. This approach implies that the functionality of the DBMS should be extended with all kinds of hard-coded distribution-specific functions. Each function simply retrieves the value(s) from the symbolic representation and computes the answer directly. This approach yields much more accurate answers with lower response times. Although this approach has major advantages, the support for non-standard continuous distributions poses a problem because distribution-specific functions for non-standard continuous distributions are not available. Then, histograms can be used for approximating non-standard continuous distributions. Another solution could be to implement the query operators, that are based on probability theory, in such way that integrals are used. In order to provide support for the different ways of computing answers for probability queries simultaneously, the DBMS should be extended with a mechanism that is able to decide whether a probability query should be executed using the symbolic form, histograms or using integrals. This mechanism should act accordingly by invoking the appropriate internal functions. This is where query plans come into play. The query plan should be chosen based on the type of continuous distributions to be queried. This approach would have an considerable positive impact on performance, accuracy as well as response time.

As we have seen in chapter 6, the performance of aggregate operators is very poor. One of the reasons is that an aggregate operation performs a cartesian product over the set of distributions and the cartesian product operation is a computational expensive operation. Maybe there is a smart way to compute an aggregated continuous distribution such that the use of an expensive cartesian product operation can be avoided. This should be further examined. Furthermore, during execution of an aggregate query operator, the values and probabilities corresponding to each discretized distribution are retrieved using identifiers. This costs a lot of overhead. Adding native support for histogram dataobjects to the underlying DBMS might have a positive effect on the performance because then the values and corresponding probabilities can be retrieved directly without having to use the complex datastructures containing identifiers.

7.3.4 Benchmarks

In chapter 6 we tried to evaluate our own Database Management System for Continuous Uncertain XML data. We assessed the performance of the system in terms of query execution time and accuracy by running self-defined benchmarks. Those benchmarks did not reflect a real-world situation (scenario): the query operators and the data sets were simple and the same. In the ideal situations, the benchmarks reflect a real-world situation and are applicable on relational as well as semi-structured data. A future direction could be to look at the Information Retrieval community. On a Text REtrieval Conference (TREC), fair comparisons are made among information retrieval systems by posing a number of questions on a set of documents, the correct answers are known in advance. Each conference produces a series of test collections, each of them consisting of a set of documents and a set of questions.

7.3.5 Support for other probability distributions

Our data model for continuous uncertain data does only have support for some continuous distributions at this moment. Only the gamma, gaussian, continuous uniform and the beta distributions are supported now. The prototype is only able to manage continuous uncertain semi-structured data that contains Gaussian distributions.

It is easy to extend the data model with other continuous distributions like the Chi, chi-square and Kent distributions etcetera. Besides that, it would be interesting to have support for discrete distributions, like Bernouilli and binomial distributions, in the data model. Although, the continuous uncertain data model does have support for discrete probabilities as it is based on the probabilistic XML data model, storing discrete distributions with its parameters could be more space-efficient. An example is the binomial distribution with parameters n and p.

7.3.6 Support for ignorance

The current data model does not take ignorance into account. The current solution is that, for missing probability mass, an empty possibility node with the remaining probability is added. This solution is used when the floor operator is applied on a continuous distribution, this operation yields missing probability mass. We foresee a solution in which missing probability mass is associated with uncovered elements. However, this requires a fundamental extension to the data model and the query language [12].

7.3.7 PRODUCT aggregate operator

As shown in section 4.3.2, the resulting distribution of the PRODUCT aggregate operator cannot be represented by a histogram. Support for this operator is desirable as motivated in section 4.3.1. Open issues with respect to supporting the PRODUCT aggregate operator reflect questions like what are the properties of the resulting (continuous) distribution, what are feasible approaches for representation and which one is the most preferable approach?

7.3.8 Updates

Updates in terms of possible worlds semantics mean that an update is performed in each possible world [12]. An update on an attribute in a probabilistic DBMS yields one of the following phenomena:

- The attribute becomes probabilistic.
  The attribute was deterministic and due to the update, the value of the attribute is modeled by a continuous distribution.
- The attribute becomes deterministic.
  The attribute was probabilistic and due to the update, the value of the attribute is a single value or a set of values. In case of a set of values, each value represents the state of the real world and is thus true (simultaneous occurrence).
- The probability that a probabilistic attribute attains a value on a particular interval becomes more likely.
  The variance is adapted by the query in such way that the area under the probability density function on that interval becomes larger.
- The probability that a probabilistic attribute attains a value on a particular interval becomes less likely.
  The variance is adapted by the query in such way that the area under the probability density function on that interval becomes smaller.

We have not paid any attention to updates on continuous uncertain data so far. Consequently, this topic needs further examination.

Bibliography

[1] Serge Abiteboul and Pierre Senellart. Querying and updating probabilistic information in XML. In *Proc. EDBT*, pages 1059–1068, Munich, Germany, March 2006.

[2] Lyublena Antova, Christoph Koch, and Dan Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, pages 606–615, 2007.

[3] Lyublena Antova, Christoph Koch, and Dan Olteanu. Maybms: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, pages 1479–1480, 2007.

[4] Omar Benjelloun, Anish Das Sarma, Anish Das Sarma, Alon Halevy, and Jennifer Widom. Uldbs: databases with uncertainty and lineage. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 953–964. VLDB Endowment, 2006.

[5] Arthur J. Bernstein and Michael Kifer. *Databases and Transaction Processing: An Application-Oriented Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[6] Patrick Bosc and J. Kaeprzyk, editors. *Fuzziness in Database Management Systems*. Physica-Verlag, 1996.

[7] Patrick Bosc and Henri Prade. An introduction to the fuzzy set and possibility theory-based treatment of flexible queries and uncertain or imprecise databases. In *Uncertainty Management in Information Systems*, pages 285–324. 1996.

[8] Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Shobhit Mathur, Chris Re, and Dan Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 891–893, New York, NY, USA, 2005. ACM.

[9] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *ICDT '01: Proceedings of the 8th International Conference on Database Theory*, pages 316–330, London, UK, 2001. Springer-Verlag.

[10] Reynold Cheng, Sarvjeet Singh, and Sunil Prabhakar. U-dbms: a database system for managing constantly-evolving data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1271–1274. VLDB Endowment, 2005.

[11] Alastair Cunningham and Christopher Jeffery. Extracting a better signal from uncertain data. *Research and analysis*, (3):363–375, 2007.

[12] Ander de Keijzer. *Management of Uncertain Data towards Unattended Integration*. PhD thesis, University of Twente, 2008.

[13] Edward Hung, Lise Getoor, and V. S. Subrahmanian. Pxml: A probabilistic semistructured data model and algebra. *icde*, 00:467, 2003.

[14] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

[15] B.R. Moole. A probabilistic multidimensional data model and algebra for olap in decision support systems. *SoutheastCon, 2003. Proceedings. IEEE*, pages 18–30, 4-6 April 2003.

[16] R.B. Moole, B.R.; Korrapati. Forecasting demand using probabilistic multidimensional data model. *SoutheastCon, 2005. Proceedings. IEEE*, pages 521–526, 8-10 April 2005.

[17] Amihai Motro. *Uncertainty Management in Information Systems: From Needs to Solutions*, section 2.2, pages 12–13. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[18] Michi Mutsuzaki, Martin Theobald, Ander de Keijzer, Jennifer Widom, Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Raghotham Murthy, and Tomoe Sugihara. Trio-one: Layering uncertainty and lineage on a conventional dbms (demo). In *CIDR*, pages 269–274, 2007.

[19] Andrew Nierman and H. V. Jagadish. Protdb: probabilistic data in xml. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 646–657. VLDB Endowment, 2002.

[20] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Representing tuple and attribute uncertainty in probabilistic databases. In *ICDMW '07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, pages 507–512, Washington, DC, USA, 2007. IEEE Computer Society.

[21] Abraham Silberschatz, Michael Stonebraker, and Jeffrey D. Ullman. Database systems: Achievements and opportunities. *Commun. ACM*, 34(10):110–120, 1991.

[22] Sarvjeet Singh, Chris Mayfield, Rahul Shah, Sunil Prabhakar, Susanne Hambrusch, Jennifer Neville, and Reynold Cheng. Database support for probabilistic attributes and tuples. In *ICDE '08: Proceedings of the 2008 IEEE International Conference on Data Engineering*.

[23] Maurice van Keulen, Ander de Keijzer, and Wouter Alink. A probabilistic xml approach to data integration. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 459–470, Washington, DC, USA, 2005. IEEE Computer Society.

[24] Mark Weiser. Ubiquitous computing. `http://www-sul.stanford.edu/weiser/Ubiq.html`.

[25] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[26] Wenzhong Zhao, Alex Dekhtyar, and Judy Goldsmith. A framework for management of semistructured probabilistic data. *J. Intell. Inf. Syst.*, 25(3):293–332, 2005.

[27] Esteban Zimányi and Alain Pirotte. Imperfect information in relational databases. In *Uncertainty Management in Information Systems*, pages 35–88. 1996.

Appendix A

Schema Continuous Uncertain XML

```
1
2  <?xml version="1.0"?>
3
4  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
5    targetNamespace="http://db.ewi.utwente.nl" xmlns="http://db.ewi.utwente.nl"
6    elementFormDefault="qualified">
7
8  <xs:element name="distribution" type="disttype"/>
9
10 <xs:complexType name="disttype">
11   <xs:all>
12     <xs:element name="symbolic" type="symbolictype"/>
13     <xs:element name="histogram" type="histogramtype"/>
14   </xs:all>
15 </xs:complexType>
16
17 <xs:complexType name="disttype_only_symbolic">
18   <xs:all>
19     <xs:element name="symbolic" type="symbolictype"/>
20   </xs:all>
21 </xs:complexType>
22
23 <xs:complexType name="symbolictype">
24   <xs:choice>
25     <xs:element name="gaussian" type="normaltype"/>
26     <xs:element name="FLOOR" type="floortype"/>
27     <xs:element name="A_MIN" type="mintype"/>
28     <xs:element name="A_MAX" type="maxtype"/>
29     <xs:element name="A_AVG" type="avgtype"/>
30     <xs:element name="A_SUM" type="sumtype"/>
31     <xs:element name="A_PRODUCT" type="producttype"/>
32   </xs:choice>
33 </xs:complexType>
34
35 <xs:complexType name="histogramtype">
36   <xs:sequence>
37     <xs:element name="h" type="xs:decimal"/>
38   </xs:sequence>
39   <xs:attribute name="left" type="xs:decimal" use="required"/>
40   <xs:attribute name="right" type="xs:decimal" use="required"/>
41   <xs:attribute name="delta" type="xs:decimal" use="required"/>
```

```
42 </xs:complexType>
43
44 <xs:complexType name="normaltype">
45   <xs:attribute name="mean" type="xs:decimal" use="required"/>
46   <xs:attribute name="variance" type="xs:decimal" use="required"/>
47 </xs:complexType>
48
49 <xs:complexType name="floortype">
50   <xs:sequence>
51     <xs:element name="distribution" type="disttype_only_symbolic"
52       maxOccurs="1" minOccurs="1"/>
53   </xs:sequence>
54   <xs:attribute name="left" type="xs:decimal" use="optional"/>
55   <xs:attribute name="right" type="xs:decimal" use="optional"/>
56 </xs:complexType>
57
58 <xs:complexType name="mintype">
59   <xs:sequence>
60     <xs:element name="distribution" type="disttype_only_symbolic" minOccurs="2"/>
61   </xs:sequence>
62 </xs:complexType>
63
64 <xs:complexType name="maxtype">
65   <xs:sequence>
66     <xs:element name="distribution" type="disttype_only_symbolic" minOccurs="2"/>
67   </xs:sequence>
68 </xs:complexType>
69
70 <xs:complexType name="avgtype">
71   <xs:sequence>
72     <xs:element name="distribution" type="disttype_only_symbolic" minOccurs="2"/>
73   </xs:sequence>
74 </xs:complexType>
75
76 <xs:complexType name="sumtype">
77   <xs:sequence>
78     <xs:element name="distribution" type="disttype_only_symbolic" minOccurs="2"/>
79   </xs:sequence>
80 </xs:complexType>
81
82 <xs:complexType name="producttype">
83   <xs:sequence>
84     <xs:element name="distribution" type="disttype_only_symbolic" minOccurs="2"/>
85   </xs:sequence>
86 </xs:complexType>
87
88 </xs:schema>
```

Listing A.1: *XML Schema of Continuous Uncertain XML data format*