

# AVATAR: Using text analytics to bridge the structured–unstructured divide

Huaiyu Zhu, Sriram Raghavan, Shivakumar Vaithyanathan  
Jayram S. Thathachar, Rajasekar Krishnamurthy, Prasad  
Deshpande

IBM Almaden Research Center  
650 Harry Road  
San Jose, CA 95120, USA

Rahul Gupta, Krishna P. Chitrapura

IBM India Research Lab  
Block I, Indian Institute of Technology  
Hauz Khas, New Delhi - 110016, INDIA

## Abstract

There is a growing need in enterprise applications to query and analyze seamlessly across structured and unstructured data. We propose an information system in which text analytics bridges the structured–unstructured divide. Annotations extracted by text analytic engines, with associated uncertainty, is automatically ingested into a structured data store. We propose an interface that is capable of supporting rich queries over this hybrid data. Uncertainty associated with the extracted information is addressed by building statistical models. We show that different classes of statistical models can be built to address issues such as ranking and OLAP style reporting. We are currently building a prototype system called **AVATAR** that utilizes an existing commercial relational DBMS system as the underlying storage engine. We present the architecture of **AVATAR** and identify several research challenges arising out of our prototyping effort.

## 1 Introduction

While traditional enterprise applications such as HR, payroll, etc., operate primarily off structured (relationally mapped) data, there is a growing class of enterprise applications in the areas of customer relationship management, marketing, collaboration, and e-mail that can benefit enormously from information present in unstructured (text) data. Consequently, the need for enterprise-class infrastructure to support integrated queries over structured and unstructured data has never been greater. To motivate the need for such an infrastructure, let us consider the following example.

### 1.1 Auto manufacturer CRM application

Consider the scenario of an auto manufacturer employing an enterprise customer relationship management (CRM) application to track and manage service requests across

ID	Model	Category	CITY	Region	Day	Comments
1	Silverado	Chevy	San Francisco	West	7/20/2002	
2	Camaro	Chevy	San Francisco	West	1/12/2002	
3	Camaro	Chevy	?	West	2/15/2001	
4	?	Buick	Los Angeles	West	9/21/2001	
5	LeSable	Buick	?	West	4/13/2001	
6	LeSable	Buick	Boston	North-East	5/27/2001	
7	Regal	Buick	NY City	North-East	3/07/2002	
8	Malibu	Buick	NY City	North-East	8/12/2002	

Table ServiceRequest

Customer replaced tires at 12000 miles at Firestone where Kevin Jackson told her that it was the rims on the Malibu that were causing the tires and brakes to fail.

Figure 1: CRM Application: A table showing customer service reports

its worldwide dealer operations. Using the CRM application, the individual dealers file “customer service reports”. Each report includes structured attributes such as “day”, “customer ID”, “make”, “model”, “dealer name”, “vehicle identification number (VIN)”, etc. In addition, there is a “comments” field where the service associate in charge of handling a service request can record additional information about the precise nature of the problem and how the issue was addressed. Figure 1 shows a simplified version of a “service reports” table and also highlights the *text* associated with one of the reports.

Using standard relational query interfaces, service reports can be queried based on the available structured attributes. Similarly, once an appropriate text-index has been constructed, reports can also be retrieved efficiently by running keyword queries over the “comments” field. However, neither of these approaches can support the following two queries:

**Query 1.** Return all the organization names starting with “Fire” that occur in service reports filed within the last 6

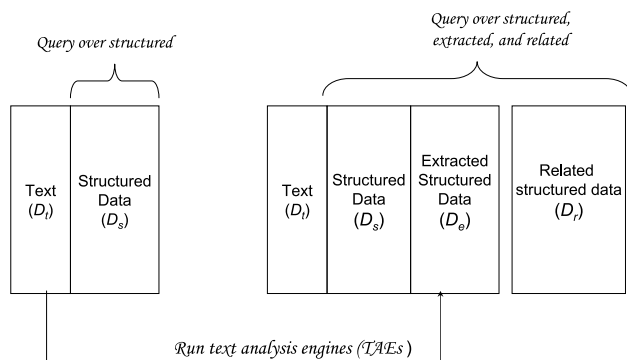


Figure 2: Bridging the structured-unstructured divide

months related to brake problems concerning Buick vehicles.

**Query 2.** What is the likelihood of brake problems in New York for Chevy vehicles whose service records contain the name “Kevin Jackson”?

Even though all of the information required to answer these queries is likely available in the table shown in Figure 1, neither of these queries can be meaningfully answered without combining information stored in the structured attributes with *semantic information embedded in text*.

## 1.2 Proposed solution

In this paper, we propose the use of *text analytics* as a mechanism to bridge this *structured-unstructured divide*. Text analytics is concerned with the identification and extraction of structured information from text. Recent advances in text analytics (see Section 1.3) have produced sophisticated techniques for analyzing free-form text to extract precisely the type of information required to answer the queries described earlier. However, for these techniques to be deployed in large enterprise-scale applications, there is a need for the data management community to play a significant role.

To this end, we envision an information system in which text analytics is used to *extract* structured information from unstructured text, the extracted information is *automatically* ingested into a structured data store, and an integrated query interface supports queries over existing structured data and the “new” extracted data. Figure 2 describes our vision through a simple schematic diagram. We assume that there is an existing information system that contains a set of text documents  $D_t$ , along with associated structured information  $D_s$ . At users’ discretion, one or more *text analysis engines (TAEs)* are used to extract structured information from the contents of  $D_t$ .

Each of these TAEs produces structured objects, called **annotations**. The set of all these annotations constitutes  $D_e$ . There may be additional structured information  $D_r$  that is somehow related to the extracted information (e.g., if  $D_e$  includes names of persons extracted from customer complaint documents,  $D_r$  can be the company’s customer

Customer replaced tires at 12,000miles, at Firestone where Kevin Jackson told her that it was the rims on the Malibu that were causing the tires and brakes to wear.

TAE	Text	Semantic Info	Probability
Named-entity mileage	12000	Mileage	0.9
Named-entity organization	Firestone	ORGANIZATION	0.85
Named-entity person	Kevin Jackson	PERSON	0.95
Topic	Document	PROBLEM=Brake	0.8

Figure 3: An example showing four annotations on a piece of text (see Section 5.1 for an explanation of the probability column)

database). Our goal is to build a new information system that can seamlessly support queries over  $D_s$ ,  $D_e$ , and  $D_r$ .

## Revisiting the CRM Example

Let us now revisit the CRM example presented earlier to understand how text analytics can be applied in this context. Using the terminology introduced above,  $D_t$  represents the set of “comments” filed by by all dealer locations.  $D_s$  would correspond to attributes such as “day”, “model”, etc., that are associated with each report.

To extract information from the “comments”, suppose the following four text analysis engines (TAEs) are executed: (i) a *named-entity PERSON TAE* trained to identify person names, (ii) a *named-entity ORGANIZATION TAE* trained to identify organizations (iii) a *MILEAGE TAE* that identifies mileage and (iv) a *topic TAE* trained to classify customer service reports based on the type of problem (“brake”, “transmission”, “engine”, “suspension”, etc.). Figure 3 shows the same service request as in Figure 1 with the corresponding set of annotations generated by the four TAEs (the probability column will be explained in Section 5.1).  $D_e$  corresponds to the set of all such annotations produced by the four TAEs for the entire comments column. Information related to these annotations present in the customer database, dealer database, service manual database, etc., constitutes  $D_r$ . Our goal is to build a system in which the information in  $D_s$ ,  $D_e$ , and  $D_r$  can be seamlessly queried and analyzed using the types of queries (Query 1 and Query 2) described earlier.

The rest of this section is organized as follows. In Section 1.3, we provide some background on the state-of-the-art in text analytics, to substantiate our claim that the time is ripe to exploit those advances in large-scale applications. In Section 1.4, we distinguish our approach to structured-unstructured integration from earlier efforts in that direction.

## 1.3 State-of-the-art in text analytics

Text analytics is a mature area of research, concerned with the problem of automatically analyzing text to extract information. Involving researchers ranging from natural language processing (NLP) to machine learning, this area has

seen tremendous growth in recent years. In particular, algorithms have been developed to perform tasks such as *entity identification* (e.g., identifying persons, locations, organizations, etc.) [10], *relationship detection* (e.g., person X works in company Y)[34, 44] and *co-reference resolution* (identifying different variants of the same entity either in the same document or different documents) [32, 36]. The popular Message Understanding Conference (MUC) is aimed at evaluating the performance of algorithms for entity detection and co-reference resolution. Best performers for entity detection were typically in the 90% precision and recall range [31].

Besides specific information extraction tasks, such as those described above, there is also significant value in classifying entire documents or large portions of a document into topic(s). NIST holds a yearly Text REtrieval Conference (TREC) that evaluates the performance of text classification algorithms. Besides TREC, papers describing text classification algorithms appear regularly in several major conferences in information retrieval, machine learning, etc. [28, 45]. Furthermore, in recognition of its significant practical potential, there are regular workshops on *Operational Text Classification Systems*[38]. A practical drawback of such classifiers is that they often require large amounts of *labeled* training data. In large, dynamic environments, obtaining large numbers of labeled data might be difficult. This bottleneck has motivated recent research in text classification to concentrate on learning from small number of labeled examples[37].

A more recent trend in NLP is detecting *affect* in documents and recently AAAI held a workshop to discuss current state-of-the-art [1]. In particular, automatic detection of *opinions* or *sentiment* (whether positive or negative) is of importance in the context of business intelligence[39]. Relevant customer reactions to a new product and new product features is available in call-center records. Extracting this information can provide valuable feedback for deciding new product features, product discontinuance etc. Consequently, results of 88% in detecting sentiment polarity from product reviews[12] is very encouraging and relevant in the context of this paper.

Despite impressive results, bar a few niche applications, advanced text analytics has not yet made a mark on mainstream enterprise software applications. We believe that it is the integration of text analytics with data management that will address this situation.

#### 1.4 The structure–precision plane

To position our approach in the context of other efforts in structured–unstructured integration, it is instructive to consider the *structure–precision* plane shown in Figure 4. As indicated in the figure, we view different classes of information systems as points in this plane. The horizontal axis of this plane represents a continuum from completely structured data (e.g., tables in a relational database) to completely unstructured data (e.g., text documents without any structured fields). The vertical axis is a continuum repre-

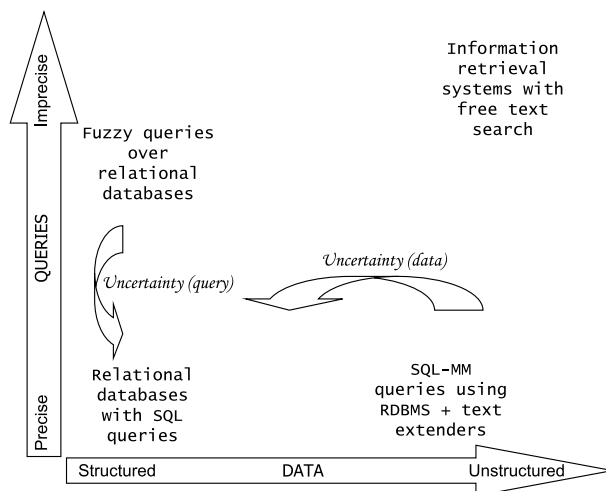


Figure 4: Structure–Precision plane

sented different levels of query precision. The bottom–left and top–right corners of this plane are occupied by traditional structured database and information retrieval systems respectively. A variety of other systems, both full-fledged products and research prototypes, can be slotted at various points in this plane. For instance, systems that incorporate unstructured text into the relational processing framework, using standards such as SQL/MM, can be placed in the (*unstructured, precise*) portion of the plane. Similarly, research prototypes such as EKS0, BANKS, DataSpot, DBXplorer, etc., [2, 27] that attempt to provide keyword or “fuzzy” queries over structured databases, are in the *structured, imprecise* part of the plane.

As illustrated by the arrows in the figure, any attempt to move down along the precision axis by converting imprecise queries to precise queries results in *query uncertainty*. Similarly, attempts to move from right to left along the structure axis by converting unstructured data to structured data results in *data uncertainty*. The presence of either data or query uncertainty (or both) leads to *result uncertainty*. In other words, precise queries over uncertain data as well as imprecise queries over certain data produce uncertain results.

While the need for unifying the separate disciplines of *information retrieval (IR)* and *databases (DB)* was recognized several years ago and engendered several research efforts in that direction (see Section 9), the vision of a fully integrated “IR–DB” systems is yet to be realized. These earlier efforts can be viewed as attempts to develop conceptual models that simultaneously handle both kinds of uncertainty.

In our approach, by restricting ourselves to movement along the structure axis, we decompose these two uncertainties and focus only on data uncertainty. We further argue that such an approach is ideally suited to meet the demands of enterprises, where, text is usually associated with a significant amount of well-organized structured information (as in our CRM example). In such environments, the ability to seamlessly query information *extracted* from text

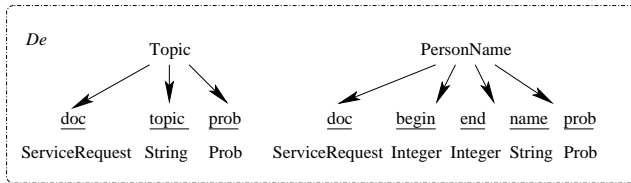


Figure 5: Complex annotation types

in conjunction with an existing body of structured information is significantly more useful than supporting an IR-style imprecise query model.

At IBM Research, we are building **AVATAR**, a prototype system based on this approach. **AVATAR** needs to interface with a system for processing, instantiating, composing, executing, and capturing the output of the individual TAEs. The NLP community has developed several software architectures, such as GATE [11, 7], ATLAS [6, 29], and UIMA [17], for this purpose. In **AVATAR**, we have implemented an interface to IBM’s UIMA architecture and allows us to leverage the large base of UIMA-compliant text analytic engines. The precise details of the interface are not relevant to this paper.

In the rest of this paper, we provide an overview of the architecture and design of **AVATAR** and enumerate several fundamental research questions arising out of our prototyping effort.

## 2 Challenges

Since the annotations produced by TAEs are structured, it might appear straightforward to simply store and query the annotations ( $D_e$ ) using a structured data store. However, there are several characteristics of  $D_e$  that preclude such a straightforward approach. We enumerate those characteristics below:

**Complex types.** Annotations produced by advanced TAEs are not merely simple atomic values, such as strings or integers, but often have a complex internal structure. However, this structure is the same for all annotations produced by a given TAE. Figure 5 shows the structure of two annotation types: the *PersonName* type produced by a *named-entity PERSON* TAE and a *Topic* type produced by a *topic* TAE. The underlined strings are attribute names and the type of a particular attribute is indicated below the name of the attribute.

**Inheritance.** Software frameworks for natural language processing (such as the UIMA framework described earlier) typically require TAEs to fit their annotation types within a type hierarchy. Such a hierarchy allows one TAE to specialize and add to the information extracted by another. Figure 6 shows an example of a set of annotation types arranged into a hierarchy. There is a root type *Annotation* that contains attributes common to all annotations. There is an annotation type *Person* that represents all annotated

person names. A subtype *Salutation-Person* represents annotation objects produced by TAEs that recognize person salutations (Mr., Mrs., Hon., etc.) and this subtype is further specialized to represent annotations of government officials (e.g., “Rep. Senator Paul Smith”) and military officials (e.g., “Lt. Colonel John Brown”).

**Variability.** A TAE typically produces one or more annotations per text document. However, given the unstructured nature of text, the number of annotations per document can vary dramatically and the variation tends to be quite specific to a given TAE. For instance, a *named-entity PERSON* TAE is likely to produce more annotation objects from a document that reports on a company’s award function as opposed to a document that represents a section of a technical product manual.

**Dynamism** It is unreasonable to expect that all “useful” semantic interpretations of a collection of text documents will be available a priori. As users and applications use the system, new TAEs will be constantly executed to extract more information from text. Furthermore, many TAEs act upon annotations produced by other TAEs to infer more complex relationships. For instance, in the CRM example, given the original documents and the annotations produced by the four TAEs listed earlier, a new TAE can infer the *works-for* relationship that associates a specific person with the company that the person works for (e.g., associate Kevin Jackson with Firestone) in Figure 3). As a result of all these factors, we expect that there can be anywhere from a few tens to several hundreds and possibly thousands of TAEs operating upon a given document collection.

**Data uncertainty.** As alluded to earlier, the characteristics of natural language are such that there is uncertainty associated with the information extracted by TAEs. There are two sources for this uncertainty.

*Algorithmic uncertainty* comes about because the particular algorithm underlying a TAE is limited in its understanding of text. For instance, given a document containing two person names, while an average human being may be able to identify both names with complete certainty, a *named-entity person TAE*, given the same task, may (i) identify both names correctly but only with 90% certainty, (ii) identify one name with 90% certainty and the other with 95% certainty, (iii) identify a piece of text incorrectly as a name with 20% certainty, and so on.

*Inherent uncertainty* is purely a consequence of the imprecise nature of natural languages. For example, given a document and asked to rate how relevant the document is to a specific topic, different individuals may respond with different ratings. Therefore, in addition to algorithmic uncertainty, a topic annotator that

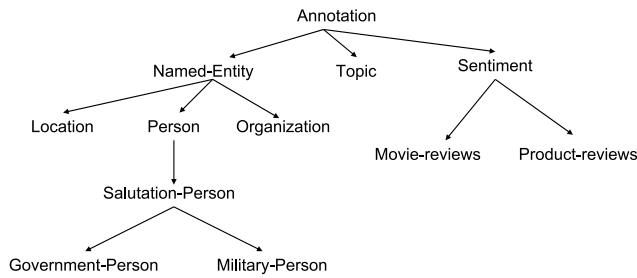


Figure 6: An example annotation type hierarchy

is asked to perform the same task will have to deal with the inherent uncertainty in the text of the document.

Given these characteristics, we are faced with the following challenges in designing a system to store and query annotations:

**Storage challenge.** Given that annotation objects have complex types, exhibit inheritance, and have highly variable statistics, the task of automatically designing efficient storage schemes to store such objects is a hard problem. For instance, naive schemes for mapping annotation objects into a relational data store will result in extremely sparse and inefficient tables (i.e., tables with a significant percentage of NULLs). In addition, because of dynamism, there will be a continuous infusion of new types and objects into the system. Designing techniques to seamlessly accommodate these new types, with minimal or no user involvement, is a significant challenge.

**Query challenge.** Due to the presence of data uncertainty, queries involving annotations can produce uncertain result sets, even if the queries themselves are precise. Based on their underlying algorithm, many TAEs automatically provide some numerical measure of this uncertainty for each annotation that they produce. The query challenge is to mathematically represent this data uncertainty and develop a statistical model that prescribes how to compute result uncertainty for a given query. The precise nature of this statistical model is application dependent. In Sections 5 and 6, we describe some possible models for point retrieval and aggregate (olap-style) queries respectively.

Many of the unique characteristics of annotations, such as large number of complex types, highly variable statistics, and dynamism, fit well within the semistructured framework. Therefore, the XML data model may seem a natural fit for storing annotations. However, there appear to be pros and cons for such an approach.

On the one hand, given the data characteristics, it appears likely that using an XML database as the back-end will alleviate some of the storage challenges. On the other hand, the fact that the schema for the annotations is well-defined, implies that we are dealing with precisely structured data, albeit hierarchical. Therefore, the underlying

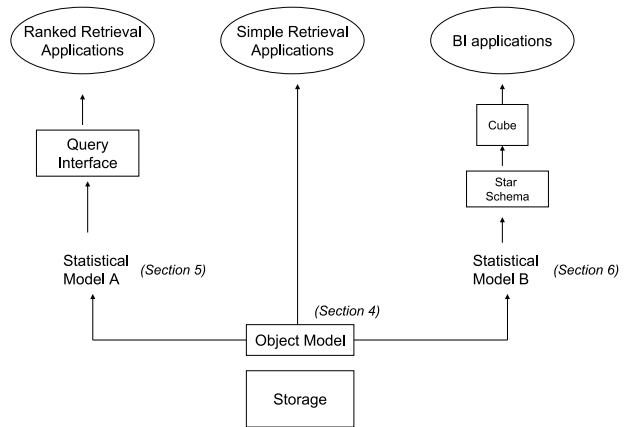


Figure 7: AVATAR application stack

database system needs to be sophisticated enough to use schema information during query optimization. Currently, we are not aware of any commercial or prototype XML database system that achieve this goal in a scalable fashion, for complex XML schema. Moreover, a large portion of structured data currently resides in relational systems. By using a relational data store as the underlying storage system for AVATAR, we avoid the problem of either having to migrate the structured data to an XML database or relying on support for querying across XML and relational data. However, since commercial database vendors are adding extensive support for XML, we intend to explore using an XML data store for AVATAR.

### 3 Architecture of AVATAR

In this section, we identify the infrastructural components for building enterprise applications in the hybrid world of structured and extracted data. Figure 7 shows the core components of our architecture, along with the sections in which those components are discussed in greater detail.

**Storage** The raw storage lies at the lowest level of the application stack. The storage component is responsible for automatically designing and populating schemes, as new complex types and objects are received from the TAEs. Depending on the underlying data store, the storage component will generate either new relational schemes, new XML schemes, or a mix of both. In our current prototype, we are using a commercial relational database as our back-end data store. However, in the interests of space, we do not present further details of the storage component in this paper.

**Object Model** The object model is an abstraction layer residing above the storage layer. Dynamism results in a back-end schema that is ever evolving. The purpose of the object model is to hide the messy details of the underlying storage, such as table and column names, while providing user-centric abstractions such as document types, annotation types, and objects. The description of the object model is agnostic to the actual

back-end storage.

**Statistical Model** To address the query challenge posed by uncertain annotations we propose the building of statistical models. The statistical model can then be used to answer queries. In this work we assume that the uncertainties are available in the form of probabilities<sup>1</sup>. Furthermore, **AVATAR** will treat the TAEs as black-boxes. Consequently, all annotation probabilities will be treated as *given* and the **AVATAR** statistical models will not attempt to capture the mechanics of the TAEs.

Certainly, the nature of the statistical models will depend on the queries and therefore the applications. We discuss below, two classes of enterprise applications roughly corresponding to OLTP and OLAP. Figure 7 provides a pictorial distinction between these applications.

**Retrieval** Broadly, retrieval applications involve the return of individual objects such as documents, annotations or more complex types. These applications can be further divided into two categories as described below.

**Simple Retrieval** Simple retrieval accesses the annotations directly (re: Figure 7). Described differently this class of applications impose no interpretations on the uncertainty associated with the annotations. Instead they are treated identical to other attributes and therefore predicates can be defined on these probabilities. A simple retrieval query based on Query 1 is as given in Query 3.

**Query 3.** *Return all the organization names starting with Fire (probability > 0.7) that occur in service reports filed within the last 6 months related to brake problems (probability > 0.6) concerning Buick vehicles.*

An alternative paradigm would be to simply *threshold* the probabilities and retain only those annotations with a probability greater than a threshold. These annotations can then be treated like any other predicates. In this paradigm the query shown in example 1 will be a return set that exceeded the thresholds.

**Ranked Retrieval** In ranked retrieval, the probabilities associated with the individual annotations are used to build appropriate statistical models. The ordering of the objects in the query result (documents or other return types) will be governed by the mechanics of the statistical model. For instance, in the query shown in example 2, the result will be an ordered list of all person names that match the structured predicates (i.e., car is a Buick and service report filed in the last 6 months). The ordering will be governed by the probabilities associated with the annotations.

---

<sup>1</sup>Uncertainty expressed in ways other than probabilities can be converted using appropriate probability models.

**Business Intelligence** The belief that aggregate information (at different levels of granularity) is important for business decisions motivated OLAP models. Specialized infrastructural support in terms of schema and join algorithms have enabled the development of large scale reporting applications. The counterpart in the structured-unstructured world is more involved. There is need for statistical models that appropriately capture the uncertainties and the relationships with dimensional hierarchies. Realizing this in a query-intensive environment while retaining the scalability of conventional OLAP is a significant challenge.

## 4 The AVATAR object model

As seen in Figure 7, the object model is a conceptual interface between the storage layer and the statistical model layer, providing mechanisms for representing the structured information extracted by text analytics. In Section 4.1, we list a set of properties that are required for representing extracted information and describe an object model that satisfies these requirements. In Section 4.2, we provide a bird's eye view of these properties using an example. A formal description of the object model is presented in Appendix A.

We would like to clarify that our object model is likely equivalent to a few other known models, such as the nested relational model (with a type system) or the object-relational model, and probably subsumed by other models such as the XML schema abstract data model. The primary reason for describing and using yet another data model is the fact that our object model is tailored for our application domain. Further, the presence of uncertainty in the annotated data implies that we require some theoretical analysis for dealing with this uncertainty. By using a simple data model that only has features required for our application domain, we hope that the theoretical analysis becomes easier.

### 4.1 Properties of the object model

As a conceptual abstraction of the underlying data, our design goals for the object model can be summarized as follows:

**Path expressions** Able to express hierarchical structures through path expressions without explicit management of foreign key references. As discussed above, such ability to shield users from unnecessary storage details is essential in a dynamic system where new types of annotation arrive at arbitrary times.

**Type constructors** Allow the construction of new types through queries, and automatically manage their storage. The results of user queries may be used to construct future queries, in at least two cases. In the first case, a user doing exploratory analysis might run queries on the system interactively, building new queries on previous ones that have promising results.

In the second case, an application might be programmed to construct queries in multiple steps.

**Document reference** Automatically keep a reference from annotations to their original documents. This allows integrated query across the structured and extracted information.

**Subtype query** Able to use subtype relations in queries. Annotations are often organized in type hierarchies. For example, an annotator that finds place names may optionally recognize the granularity, such as country or city. A query for place names should therefore be able to retrieve annotations that are explicitly places, as well as those that are countries and cities.

The object model can be formalized as a system of types and objects. In the example in Section 1.1, each row in the CRM table is regarded as a document. The collection of documents forms a type  $D$ , each individual document  $d$  in the collection being an object of that type. Running a TAE on a document  $d$  produces zero or more annotations  $a$ . The collection of such annotations defines a new type  $A$ , each annotation  $a$  being an object of that type.<sup>2</sup> Apart from attributes having usual semantics, each annotation  $a$  also has an attribute  $a.d$  referring to the original document  $d$  from which it is obtained. This makes it possible to query both structured, unstructured and extracted information simultaneously.

One distinguishing feature of the object model is that each attribute of an object is automatically an object, and each attribute of a type is automatically a type. This allows chained attribute references in queries. For example, the object model permits expressions such as

`car.owner.name like 'John%'` (1)

Another important characteristic of our object model is that the subtype relation is part of the intrinsic semantics. This means that, for example, if `Car` is a subtype of `Vehicle`, then a query for `Vehicle` would retrieve all objects of `Car` as well as other subtypes of `Vehicle`.

## 4.2 Example queries in the object model

In this subsection we illustrate various concepts in the object model with an example. The example may look somewhat contrived, due to our desire to fit all relevant concepts into the same example. However, in real world applications, any combination of the issues discussed here can appear.

Let us revisit the CRM example in Figure 1. The table shown in the figure can be viewed as the definition of a type, `ServiceRequest`. Each type in our theory is defined by its schema and its set of objects. The schema of a type is a mapping from its set of attribute names to attribute types, as shown in Figure 8, where the type of an attribute is written under the name of the attribute. For example, `city` is

an attribute of type `City`. The set of objects for this type is defined by the rows in the table shown in 1.

A user runs the *topic TAE* on the `comments` attribute of `ServiceRequest`, producing an annotation type `Topic` (Figure 8). It contains attributes `topic` (name of the topic), `prob` (probability of the text being on topic) and `doc`, which is a reference to the original document. Such references are automatically maintained by the system.

As shown in the same figure, the user can also run a *named-entity PERSON TAE* on the `comments` attribute of `ServiceRequest` to produce an annotation type `PersonName`. The `begin` and `end` attributes indicates the location of the named entity in the text given in terms of character offsets.

A simple retrieval query for the combined data  $D_s + D_e$  could be:

```
for all
  ServiceRequest r,
  Topic t,
  PersonName p,
where
  r = t.doc = p.doc and
  r.make like 'Chev%' and
  t.topic = 'brake' and
  t.prob > 0.5,
return PersonTopic
  doc r, topic t, person p
```

This query is quite similar to a SQL query. However, since attributes of types are again types, it is allowed that the query returns a list of complete objects. This can be viewed as defining a new type `PersonTopic`, as shown in Figure 8. The schema of this type is defined by the return statement of the query. The object set of the query consists of all the tuples  $(r, t, p)$  that satisfy query. Once defined, such a type can be used by the system just like any other type.

In applications, it is often useful to link the extracted information to some related information contained in existing database. In our current example, the person name extracted from the `comments` field is often the name of a CRM representative. These names can be correlated to names in the `Employee` database. A *same-person TAE* can be run on the `PersonName` type and `Employee` type, producing a new type `SamePerson`, which contains two attributes pointing to the extracted `PersonName` and `Employee` types (Figure 8). The system handles subtype relations transparently: the *same-person TAE* is written for a pair of `Person` types, yet it can be run on the pair of `PersonName` and `Employee` types, which are subtypes of type `Person`.

These newly defined types allow subsequent queries to be handled much simpler both in terms of user input and in terms of performance, since they can be based on the types constructed in previous queries. For example, the query

```
for all
  PersonTopic pt,
  SamePerson sp,
```

<sup>2</sup>A TAE can be written to produce objects of more than one type. However this detail will not affect what is being discussed here.

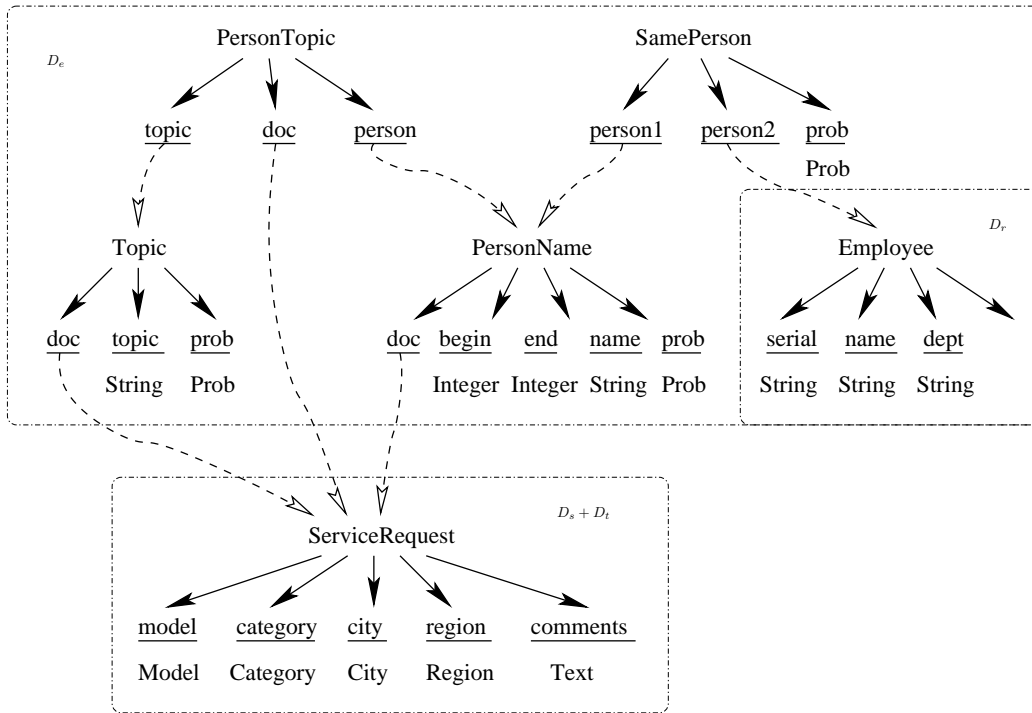


Figure 8: Example schema of documents and annotations

where

```

pt.person = sp.person1 and
sp.prob > 0.9
return
pt.doc.model, sp.person2.dept

```

would return all those (model, dept) pairs where a person likely (more than 90% chance) of that department has handled a service request about probably (more than 50% chance) a brake problem on the specific Chevy model. This example demonstrates the convenience of using path expressions in queries without explicit foreign key reference.

It is worth pointing out that neither the query syntax nor the back-end storage of the object model are tied to the relational view. In Appendix A.1, the query is expressed in a more abstract form that can be translated into any specific query language. In Appendix A.3, a simple mapping to a relational back-end is outlined, as is currently being implemented.

## 5 Ranking

As shown in Figure 7, in ranked retrieval, a statistical model prescribes how to compute result uncertainty from data uncertainty. Once an uncertainty measure has been associated with each object in the result set, the result set can be sorted in decreasing order of this measure to produce a ranked/ordered list of result objects. In this section, we describe a framework for building such statistical models for ranked retrieval. We will begin by developing a probabilistic interpretation for the uncertainty associated with an annotation.

### 5.1 Interpreting annotation uncertainties

As mentioned earlier in Section 4, for the purposes of developing a statistical model, we treat TAEs as black boxes and make no attempt to precisely model how a TAE computes its uncertainty measure. In particular, we treat the numerical uncertainty measures provided by TAEs as *probabilities*, irrespective of whether these measures were indeed generated by a probability model.

We will assume that each annotation contains a single probability. Without loss of generality, assume that this probability is always stored in the special attribute prob. When a TAE produced an annotation object, it produces both a type assignment (i.e., a statement that an object belongs to a particular type) and a set of attribute values. Therefore, the uncertainty in an annotation can either be in the type assignment (i.e., whether the annotation truly belongs to the specified type) or in one of the attribute values. To illustrate, consider the Topic annotation type shown in Figure 8. We know that when a TAE produces objects of this type, the uncertainty is only in the value of the attribute Topic.topic. On the other hand, when a *named-entity PERSON* TAE produces an object  $a$  of type PersonName, the uncertainty is in whether that object is indeed a person name, i.e., in whether  $a$  belongs to type PersonName.

For the case when uncertainty is in the attribute, we posit the following interpretation:

**Definition 1 (Accuracy statement).** *Associated with every annotation type  $A$ , we have an attribute  $x_A$  that is uncertain. Let  $r(a, x_A)$  denotes a random variable that represents the “true” value of  $a.x_A$  for every annotation  $a$  of*

type  $A$ . Then, the statement  $(r(a, x_A) = a.x_A)$  is called the accuracy statement associated with annotation  $a$ . We will use  $m_a$  to denote the accuracy statement associated with  $a$ .

**Definition 2 (Annotation probabilities).** If  $a$  is an annotation of type  $A$  whose uncertainty value is stored in  $a.\text{prob}$ , we have

$$P(m_a|d) = a.\text{prob} \quad (2)$$

$$P(\text{NOT } m_a|d) = 1 - a.\text{prob} \quad (3)$$

$$P(r(a, x_A) = v|d) = 0 \quad \forall v \neq a.x_A \quad (4)$$

In other words, we interpret  $a.\text{prob}$  as the probability that the value of  $a.x_A$  is accurate for annotation  $a$ . Furthermore, we assume that the probability that the true value of  $a.x_A$  is anything else is zero.<sup>3</sup>

As an example, if  $A$  is the `Topic` annotation shown in Figure 8, we have  $x_A = \text{topic}$ . Thus, given a document  $d$  and a topic annotation  $t$  with  $t.\text{prob} = 0.6$  and  $t.\text{topic} = \text{"Brake"}$ , we have

$$P(r(t, \text{topic}) = \text{Brake}|d) = 0.6 \quad (5)$$

$$P(r(t, \text{topic}) = v|d) = 0, \quad \forall v \neq \text{Brake} \quad (6)$$

Uncertainty in the type assignment can always be modeled in terms of uncertainty in an attribute by introducing a special attribute named `type`. For instance, when  $A = \text{PersonName}$ , we can set  $x_A = \text{type}$  and implicitly set  $r(a, \text{type}) = \text{PersonName} \forall a \in \text{PersonName}$ . Thus, when a *named-entity* `PERSON` TAE produces a `PersonName` object  $a$  with  $a.\text{prob} = 0.7$ , the interpretation is that with 70% probability,  $a$  represents a person name (and hence is of type `PersonName`).

For convenience, in the rest of this section, we will use  $\hat{p}(a|d)$  to denote the probability associated with annotation  $a$ . Thus,

$$\hat{p}(a|d) = P(m_a|d) = a.\text{prob}$$

## 5.2 Probability model

Using the above interpretation of annotation uncertainties, we will now develop a simple probability model for documents and annotations and present an associated ranking scheme. To make our task tractable, we make the following assumptions:

**Assumption 1.** We will only focus on document ranking in this section. In other words, we will assume that every query returns a collection of documents. The task of generating ranked sets for queries that return annotations and other complex types poses greater challenges that we discuss in Section 8.

<sup>3</sup>The third equation in Definition 2 is introduced for a very specific reason for the subsequent development of the ranking model. Details are provided in [16].

**Assumption 2.** We will only consider document types and annotation types produced by TAEs that directly operate on documents. Annotations produced by TAEs that act upon other annotations or types produced through user-defined queries will not be modeled in this section. Thus, for the example shown in Figure 8, we will only deal with types `ServiceRequest`, `PersonName`, and `Topic`.

**Assumption 3.** We will assume that there is only one annotation object per document for every annotation type. While this condition is true for the `Topic` annotation type, it is not in general true for `PersonName`, since there may be several person names in the same document. However, probabilistically modeling set-valued annotation types is a complex task that is reserved for future research.

Given these assumptions, we associate with a document type  $D$ , a set of structured attributes  $\mathcal{S}(D) = \{s_1, s_2, \dots, s_k\}$  and a set of extracted attributes  $\mathcal{E}(D) = \{e_1, e_2, \dots, e_r\}$ . Each extracted attribute represents an annotation object. For instance, in Figure 8, for the document type  $D = \text{ServiceRequest}$ , we have  $\mathcal{S}(D) = \{\text{model}, \text{category}, \text{city}, \text{day}, \text{region}\}$  and  $\mathcal{E}(D) = \{\text{topic}, \text{personName}\}$ .

## 5.3 Probabilistic ranking model

Given a query  $q(D)$  where  $D$  is a document type and  $q$  is a predicate involving the attributes of  $D$ , the result of  $q(D)$  is the set of documents of type  $D$  that satisfy  $q$ . In the result, a document is ranked based on the probability that it satisfies the query predicate. In other words, the rank of document  $d$  is given by

$$\text{rank}(d) = P(q(d)|d) \quad (7)$$

Let  $X_q$  denote the attributes of  $d$  that are relevant to  $q$  and let  $M_q$  denote the *accuracy statements* of the extracted attributes of  $d$  that are relevant to  $q$ . For instance, if the query is “return all documents where `model = Buick` and `topic = Brake`”,  $X_q = (\text{model}, \text{topic})$  and  $M_q = \{m_{d.\text{topic}}\}$ . We can rewrite (7) as

$$\text{rank}(d) = P(q(d)|X_q, M_q, d)P(X_q, M_q|d) \quad (8)$$

The first term on the right hand side of the above equation represents the probability that a particular document will satisfy the query predicate, given the values of all the relevant attributes of the document. Clearly, this term corresponds to a precise database query with a deterministic 1/0 answer. In particular, if we restrict ourselves to documents which actually satisfy the query predicate  $q(d)$ , the first term resolves to unity and  $\text{rank}(d)$  reduces to  $P(X_q, M_q|d)$ . Since  $X_q$  is a deterministic variable representing the attributes of  $d$ , we get:

$$\text{rank}(d) = P(M_q|d) \quad (9)$$

Thus, we have reduced the problem of assigning a rank for document  $d$  to the problem of estimating the probability  $P(M_q|d)$ .

A simple approach for estimating the probability in (9) is to make the assumption that every annotation on a document is independent of all other annotations. We can therefore decompose  $M_q$  into individual terms involving each annotation. Thus, (9) becomes:

$$\text{rank}(d) = \prod_{e \in X_q \cap \mathcal{E}(D)} P(m_e|d)$$

Finally, using (5.1), we get

$$\text{rank}(d) = \prod_{e \in X_q \cap \mathcal{E}(D)} \hat{p}(e|d)$$

## 6 Business Intelligence

As in business intelligence, with structured data the goal in the structured-unstructured world is to enable slicing and dicing of measures with respect to dimensions. *Measures* and *dimensions* in the world of  $D_s + D_e$  are similar to that of the structured world. As an example consider the Query 2 in Section 1.1. Certainly, the answer to this query is the aggregate *Brake information* of all service records containing the Person name *Kevin Jackson* reported from *New York* for *Chevy* vehicles. Complications arise due to the fact that the measure (Brake information) is a probability distribution and one of the dimensions (Person annotation) has a probability associated with it. To enable a systematic tackling of the issues we consider the following three cases.

### Case 1: Dimensions from $D_e + D_s$ and measure from $D_e$

Consider Query 2. This is the most general of the cases we consider.  $D_e$  contributes both to the measure and the dimension of the cube from which this query is answered.

### Case 2: Dimensions from $D_e + D_s$ and measure from $D_s$

Consider, instead, the query given below. Person is an “extracted” dimension while the measure is simply the count of number of service records.

**Query 4.** *How many service records contain the name “Kevin Jackson” and are from New York ?*

### Case 3: Dimensions from $D_s$ and measure from $D_e$

This is the simplest of the three cases where Location and Automobile are the dimensions and the measure is the extracted “Brake topic”.

**Query 5.** *What is the likelihood of brake problems in New York for Chevy vehicles ?*

The cases described above are listed in decreasing order of complexity. Cases 1 and 2 need to deal with the probabilities as well as the representational issues - i.e., we need the

equivalent of a star-schema for dimensions extracted from  $D_e$ . The representational issue arise because the data in  $D_e$  may not be regular (e.g., the mentions of Kevin Jackson may be highly variable across documents). Fitting such irregular data into a star-schema is challenging. A related issue is the definition of hierarchies for dimensions extracted from  $D_e$ . These could be defined using the annotation type hierarchy (Figure 6) or possibly from  $D_r$  (re:Section 1).

Case 3, on the other hand, needs simply to deal with probabilistic measures. It is precisely for this case that we have proposed a solution which is described in our recent submission[8]. Probabilistic OLAP (PrOLAP), based on theoretical development described in [20], proposes a statistical model-based solution. Further, the query in Case 3 above is interpreted as  $P(\text{Topic} = \text{Brake} | \text{City} = \text{NY}, \text{Category} = \text{Chevy})$ . The goal is to answer all such queries (slice and dice) preferably using existing SQL support for OLAP applications. A detailed explanation of PrOLAP is provided in our recent submission [8] but an overview is provided in Appendix B.

## 7 Application Examples

The commercial success of relational databases is primarily due to large scale applications such as enterprise resource planning (ERP), human resources (HR), payroll, OLAP reporting, etc. Similarly, the success of a system such as **AVATAR** hinges largely on the identification of *killer applications* that can leverage the combined structured-unstructured data. The first important step is to identify applications that contain data sources where text is associated with important structured data. Besides CRM, other such hybrid sources are e-mail and documents in collaboration applications such as Lotus Notes. We provide below a few scenarios that drive home the value of **AVATAR**.

**CRM** Revisiting our CRM example let us consider a sales promotion application. Such a promotion could have one or more of several possible goals: customer retention, increasing goodwill, reduce inventory, etc. The application designer would translate this broad goal into appropriate queries on **AVATAR**. Existing systems *score* customers while playing a delicate balance between precision (not miss customers who are likely to buy) and recall (not mail rebates to customers who might treat it as spam). A system such as **AVATAR** incorporates a new dimension and therefore can potentially increase the timeliness and the precision. Consider, for example, an application that sends out customer rebate coupons. The following query helps to target such rebates towards customers that drive older Buicks and have had recent engine problems. The results of the query could be processed automatically to send out the offers.

**Query 6.** *Return a list of customers who drive Buicks manufactured before 1998 and have complained of engine problems in the last 6 months.*

Alternatively, consider an application trying to track problem types that required the attention of “Service Managers” before the record is closed. This requirement can be translated to the following query.

**Query 7.** *Return all problems from service records that mention a Service Manager and the record has been closed ?*

**E-Mail** Another important domain for seamless querying across structured and unstructured data is e-mail. Consider a user who receives regular emails from John Smith on various topics. The search problem is where the user is looking for the name and phone number of a particular database expert that John had referred to him in an e-mail. This search need is captured, somewhat, in the query given below. In the absence of annotations, the only recourse is to perform some sort of keyword search on emails from John Smith and then read every email in the result set to obtain the information.

**Query 8.** *Return all phone numbers and names of Persons mentioned in e-mails from John Smith that discuss database research ?*

The above query assumes that the only annotations available are named-entity (Persons) and topics. Suppose, however, a relationship annotator was available and indeed had identified Persons as database researchers then the query would be somewhat different.

**Query 9.** *Return all names and phone numbers of Persons who are database experts mentioned in John Smith’s emails ?*

Note that Query 9 is a much stronger query than Query 8 but requires a significantly more powerful relationship identification TAE. However, as shown by Query 8 even simple TAEs, available today, can be used to form very powerful queries.

## 8 Issues for Future Research

The vision and architecture that we have presented in this paper raises several issues that warrant significant research. Drawing from our experience in building the **AVATAR** prototype, we enumerate some of these issues below.

### *Time-dependent object model*

The object model prescribes a set of rules for a consistent system of types and objects. In practice the type system is ever expanding. New types may be added due to annotations or queries. New subtype relations may be introduced. The set of objects for a type may change due to new annotations. Since these types and objects are automatically managed by the system, it is important to develop rules by which the consistency of the system is maintained at each

step. Since each attribute is a reference, the distinction between copy and reference semantics becomes an issue for new types defined by user queries. In essence, it is necessary to develop a *theory of time-dependent type system*.

### *Multiple annotators for the same type*

In a real world situation, it is often the case that several versions of TAEs for essentially the same semantic information may be available. They may be developed independently, using similar or different algorithms. The types of annotations they produce may be different, with different but often overlapping attributes. A user can apply several such TAEs on the same documents, producing slightly different annotations. The user might want to consider all of them as subtypes of a supertype in some queries, yet as separate types in other queries. The user might even want the system to discover possible subtype relations automatically. Correct treatment of these issues require careful investigation.

### *Extended ranking models for retrieval*

In Section 5, we made several assumptions about the interpretations of annotation uncertainties. We assumed that query return types are documents, that queries only involve direct annotations on documents (rather than annotations on annotations or annotations defined by user queries), that each document contains exactly one annotation of a given type, and that the probability for each annotation is independent of other annotations. Removing these restrictions would enlarge the set of query semantics for which ranking of the results can be defined. However, removing any one of these assumption will remove independence among the probability distribution for different annotations. For example, if an annotation  $p$  points to another annotation  $b$  with probability, there will be statistical dependence between these two annotations.

### *Extended models for business intelligence*

The Our current effort in OLAP has been restricted to the simple case where only the measure is derived from  $D_e$ . Extensions to incorporate extracted dimensions into the OLAP data-model is a direction for future research. The present theoretical basis for PrOLAP is restricted to obtaining *average* aggregate behavior over distributions[20]. Extending the analysis to facilitate obtaining extreme and other aggregate behaviors is another future area of research. Data paucity, observations for only some cells of a cube is available, is another important issue that needs addressing. Unlike conventional OLAP, the probabilistic model in PrOLAP provides a predictive capability that enables a systematic solution.

## 9 Related work

As we mentioned in Section 1, there is a long history of work in the area of bridging text retrieval systems and structured databases [14, 22, 13, 23, 43, 9, 30, 19, 15, 40].

A number of research prototypes have been developed to address the problem of supporting keyword queries over structured data (the structured, imprecise part of the plane in Figure 4). In DBXplorer [2] and DISCOVER [27], techniques for supporting keyword queries over relational databases are presented. In [26], algorithms for returning the top-k results in this context are presented. In [5, 21], algorithms for evaluating keyword queries over graph-structured data are presented.

Recently, in [3, 4, 18, 41, 42], query languages that integrate information retrieval related features such as ranking and relevance-oriented search into XML queries have been proposed. Techniques to evaluate these ranked queries are also proposed in [3, 4, 41, 42]. In [33], the problem of ranking SGML documents using term occurrences is considered.

Several proposals have been made for ranked search over a corpus of document databases combining keyword and structure components [25, 35]. In [25], structure is imposed on text documents by partitioning the text into multi-paragraph units that correspond to subunits. The structure part of the query can restrict the keywords to a particular subtopic. In [35], a more general structure model is presented, where the structure of the text document is organized as a set of independent hierarchies.

In [24], an approach is presented for adding structure to the (unstructured) HTML web pages present on the World Wide Web. The authors argue that creating structured data typically requires technical expertise and substantial upfront effort; hence people usually prefer to create unstructured data instead. So, one of the core ideas of the system is to make structured content creation, sharing, and maintenance easy and rewarding. They also introduce a set of mechanisms to achieve this goal. Notice how this approach is different from our approach, where we use text analytics to add structure to unstructured data. Our focus is on enterprise applications, whose characteristics are a lot different from web data authoring. First, domain knowledge enables us to use a variety of text analytic tools. In contrast, in [24] the authors mention that they do not use information extraction techniques in their system as it will require domain knowledge, which may not be available for their scenario. Second, the cost for manually adding structure to unstructured data is likely to be pretty high in an enterprise application.

## 10 Conclusion

In this paper, we have laid out a case for text analytics as a mechanism for bridging the structured-unstructured divide in the enterprise. We presented an overview of recent advances in text analytics and argued that the data management community has a significant role to play in bringing these advances to the enterprise. Towards this end, we laid out a vision and architecture for how these two communities can come together. Based on our prototyping experience, we identified several challenges as well as open issues that warrant further investigation. We believe that the

vision that we have laid out in this paper will prove to be a fertile area of research with contributions from the machine learning, data management, and NLP/IR communities.

## References

- [1] AAAI 2004. <http://www.clairvoyancecorp.com/Research/Workshops/AAAI-EAAT-2004/home.html>.
- [2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proc. of ICDE*, 2002.
- [3] S. Al-Khalifa, C. Yu, and H. V. Jagadish. Querying structured text in an xml database. In *SIGMOD*, 2003.
- [4] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *EDBT*, 2002.
- [5] G. Bhalotia et al. Keyword searching and browsing in databases using BANKS. In *Proc. of ICDE*, 2002.
- [6] S. Bird, D. Day, J. Garofolo, J. Henderson, C. Laprun, and M. Liberman. ATLAS: A flexible and extensible architecture for linguistic annotation. In *Proc. of the Second Intl. Language Resources and Evaluation Conf. (LREC)*, 2000.
- [7] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to meet new challenges in language engineering. *Natural Language Engineering*, June 2004.
- [8] Doug Burdick, Prasad Deshpande, T.S. Jayram, and Shivakumar Vaithyanathan. Prolap: Probabilistic olap, 2004.
- [9] W. Bruce Croft, Lisa Ann Smith, and Howard R. Turtle. A loosely-coupled integration of a text retrieval system and an object-oriented database system. In *Proc. of the 15th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 223–232, June 1992.
- [10] H. Cunningham. Information extraction - a user guide. Technical Report CS-97-02, University of Sheffield, 1997.
- [11] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust nlp tools and applications. In *Proc. of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL02)*, 2002.
- [12] D. Dave and S. Lawrence. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In *Proc. of the Twelfth Intl. World Wide Web Conference (WWW2003)*, 2003.

- [13] Arjen P. de Vries and Annita N. Wilschut. On the integration of IR and databases. In *Proc. of the 8th IFIP 2.6 Working Conference on Database Semantics*, January 1999.
- [14] Samuel DeFazio, Amjad M. Daoud, Lisa Ann Smith, Jagannathan Srinivasan, W. Bruce Croft, and James P. Callan. Integrating IR and RDBMS using cooperative indexing. In *Proc. of the 18th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 84–92, July 1995.
- [15] Stefan Deßloch and Nelson Mendonça Mattos. Integrating SQL databases with content-specific search engines. In *Proc. of 23rd Intl. Conf. on Very Large Data Bases*, pages 528–537, August 1997.
- [16] Huaiyu Zhu et. al. Probabilistic ranking models for annotated data. Technical report, IBM Research Technical Report, 2004.
- [17] D. Ferrucci and A. Lally. UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, June 2004.
- [18] N. Fuhr and K. Grobjochn. XIRQL: A language for information retrieval in XML documents. In *Proc. of SIGIR*, 2001.
- [19] Norbert Fuhr and Thomas Rolleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1):32–66, 1997.
- [20] Ashutosh Garg, Jayram Thathachar, Shivakumar Vaithyanathan, and Huaiyu Zhu. Generalized opinion pooling, 2004.
- [21] R. Goldman et al. Proximity search in databases. In *Proc. of VLDB*, 1998.
- [22] David A. Grossman, Ophir Frieder, David O. Holmes, and David C. Roberts. Integrating structured data and text: A relational approach. *Journal of the American Society for Information Sciences*, 48(2):122–132, 1997.
- [23] J. Gu, U. Thiel, and J. Zhao. Efficient retrieval of complex objects: Query processing in a hybrid db and ir system. In *Proc. of the 1st German National Conf. on Information Retrieval*, 1993.
- [24] Alon Y. Halevy, Oren Etzioni, AnHai Doan, Zachary G. Ives, Jayant Madhavan, Luke McDowell, and Igor Tatarinov. Crossing the Structure Chasm. In *CIDR*, 2003.
- [25] M. Hearst and C. Plaunt. Subtopic structuring for full-length document access. In *Proc. of SIGIR*, 1993.
- [26] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, 2003.
- [27] V. Hristidis and Y. Papakonstantinou. DISCOVER: keyword search in relational databases. In *VLDB*, 2002.
- [28] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proc. of 10th European Conf. on Machine Learning (ECML98)*, 1998.
- [29] C. Laprun, J. Fiscus, J. Garofolo, and P. Sylvain. A practical introduction to ATLAS. In *Proc. of the Third Intl. Conf. on Language Resources and Evaluation (LREC)*, 2001.
- [30] Clifford A. Lynch and Michael Stonebraker. Extended user-defined indexing with application to textual databases. In *Proc. of the Fourteenth Intl. Conf. on Very Large Data Bases*, pages 306–317, August 1988.
- [31] E. Marsh and D. Perzanowski. Overview of results of the muc-7 evaluation. In *Proc. of the Sixth Message Understanding Conf. (MUC-7)*, pages 13–31, 1996.
- [32] Joseph F. McCarthy and Wendy G. Lehnert. Using decision trees for coreference resolution. In *IJCAI*, pages 1050–1055, 1995.
- [33] S. Myaeng et al. A flexible model for retrieval of SGML documents. In *SIGIR*, 1998.
- [34] Kambhatla Nanda. Combining lexical, syntactic and semantic features with maximum entropy models for extracting relations. In *Proc. of the 42nd Anniversary Meeting of the Association for Computational Linguistics (ACL04)*, 2004.
- [35] G. Navarro and R. Baeza-Yates. Proximal nodes: A model to query document databases by content and structure. *ACM Transactions on Information Systems*, 15(4), 1997.
- [36] Vincent Ng and Claire Cardie. Improving machine learning approaches to coreference resolution. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, 2002.
- [37] Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.
- [38] OTC 2001. <http://www.daviddlewis.com/events/otc2001/index.html>.
- [39] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? Sentiment classification using machine

learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86, 2002.

- [40] Ron Sacks-Davis, Alan J. Kent, Kotagiri Ramamohanarao, James A. Thom, and Justin Zobel. Atlas: A nested relational database system for text applications. *Transactions on Knowledge and Data Engineering*, 7(3):454–470, 1995.
- [41] T. Schlieder and H. Meuss. Result ranking for structured queries against XML documents. In *DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries*, 2000.
- [42] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *EDBT*, 2002.
- [43] Marc Volz, Karl Aberer, and Klemens Bohm. An OODBMS-IR coupling for structured documents. *Bulletin of the Technical Committee on Data Engineering*, 19(1):34–42, 1996.
- [44] Aone Chinatsu Zelenko Dmitry and Richardella Anthony. kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106, 2003.
- [45] Tong Zhang and Frank J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4(1):5–31, 2001.

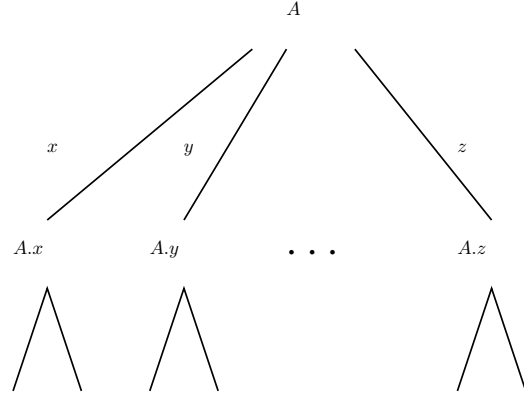


Figure 9: Type attribute paths as a tree

## A Outline of Object Model

### A.1 The type system of object model

Here we give an outline of the object model. This subsection does not deal with documents and annotations, which will be treated in the next subsection. The basic concepts of concern here are objects  $a, b, c, \dots$ , types  $A, B, C, \dots$ , and attribute names  $x, y, z, \dots$ . A type  $A$  can be a subtype of  $B$ , written as  $A \preceq B$ . The subtype relation forms a partial order among the types.

Every object  $a$  belongs to a specific type  $\mathbf{type}(a)$ , and every type  $A$  is associated with a set of objects  $\mathbf{objs}(A)$ , consisting of all the objects that are of subtype of  $A$ . That is,

$$a \in \mathbf{objs}(A) \iff \mathbf{type}(a) \preceq A. \quad (10)$$

As a consequence of  $\preceq$  being a partial order, we have

$$A \preceq B \implies \mathbf{objs}(A) \subseteq \mathbf{objs}(B). \quad (11)$$

Every type  $A$  has a set of named attributes. The schema of  $A$  maps each attribute name  $x$  to corresponding attribute type,  $A.x$

$$\mathbf{sch}(A) : x \rightarrow A.x. \quad (12)$$

If  $A \preceq B$ , then  $A.x \preceq B.x$  for every attribute  $x$  of  $B$ . Similarly, every object  $a$  has a set of named attributes. Each attribute name  $x$  maps to an object  $a.x$ . If  $\mathbf{type}(a) \preceq A$ , then  $\mathbf{type}(a.x) \preceq A.x$  for every attribute  $x$  of  $A$ .

Since  $A.x$  is again a type whenever it is defined, we can chain the attributes to denote types such as  $A.x_1.x_2.\dots$ . We call  $\mathbf{x} = x_1.x_2.\dots.x_n$  a path. The set of all valid paths for type  $A$  is denoted  $\mathcal{C}(A)$ .

As shown in Figure 9, a type  $A$  can be associated with a rooted edge-labeled tree. The root of the tree is the type  $A$ . Each label  $x$  on an edge leaving  $A$  is an attribute, leading to the child node  $A.x$ . An edge path  $x_1, x_2, \dots, x_m$  starting from the root  $A$  is a path  $x_1.x_2.\dots.x_m \in \mathcal{C}(A)$ .

As in relational algebra where queries are expressed as table constructors, in our object model queries are expressed as type constructors. These type constructors are

similar to the corresponding operators in relational algebra. The main differences are: (1) Instead of dealing with a single attribute (a column in a table), we deal with an entire path of attributes. (2) Subtypes relations are taking into consideration.

**Cartesian product.** The Cartesian product type constructor  $T$  forms a new type  $A$  by stringing together several existing types  $A_i$  using attribute names  $x_i$ . Given a set of names  $x_i$  and corresponding types  $A_i$ , the type  $A$  is defined such that  $A.x_i = A_i$ , and that  $\mathbf{objs}(A)$  consists of objects  $a$  such that  $a.x_i = a_i$  and  $a_i \in \mathbf{objs}(A_i)$ . We write

$$A = T S, \quad \text{where } S : x_i \rightarrow A_i. \quad (13)$$

**Projection.** The projection operator forms a new type  $B$  by stringing together a subset of paths  $y_i$  of a type  $A$  using attribute names  $x_i$ . Given a type  $A$ , a set of names  $x_i$  and corresponding paths  $y_i \in \mathcal{C}(A)$ , the type  $B$  is defined such that  $B.x_i = A.y_i$ , and that  $\mathbf{objs}(B)$  consists of objects  $b$  such that  $b.x_i = a.y_i$  and  $a \in \mathbf{objs}(A)$ . We write

$$B = \pi_s A, \quad \text{where } s : x_i \rightarrow y_i \quad (14)$$

**Selection.** The selection operator forms a new type  $B$  by selecting a subset of objects of a type  $A$  according to a predicate  $p$  defined on  $A$ . A predicate  $p$  defined on a type  $A$  is a map from objects of  $A$  to the truth values. Given a type  $A$  and a predicate  $p$  defined on  $A$ , the type  $B$  is defined such that it has the same schema as  $A$ , and that  $\mathbf{objs}(B)$  consists of objects  $a \in \mathbf{objs}(A)$  that satisfies the predicate,  $p(a)$ . We write

$$B = \sigma_p A, \quad (15)$$

Note that the predicate  $p$  may also involve path expressions.

**General type constructor.** A general type constructor can be formed using Cartesian product, projection and selection operators. Given a finite map  $f$  from names to types, a finite map  $s$  from names to paths in  $\mathcal{C}(T f)$ , and a predicate  $p$  defined on the type  $T f$ , a new type can be defined as

$$\pi_s \sigma_p T f. \quad (16)$$

This is a general form for specifying queries in the object model. It is written in a form similar to relational algebra. To bring out the conceptual linkage between the object model with existing data models, it is instructive to write this in syntaxes similar to these other formalisms. Thus we have, in the style of relational calculus or set comprehension notation,

$$\{a.s : a \in f, p(a)\}, \quad (17)$$

in the style of SQL,

$$\text{SELECT } s \quad (18)$$

$$\text{FROM } f \quad (19)$$

$$\text{WHERE } p \quad (20)$$

and finally, in the style of XQuery,

$$\text{for } a \text{ in } f \quad (21)$$

$$\text{where } p(a) \quad (22)$$

$$\text{return } a/s \quad (23)$$

The syntactic details of these query forms are of course different, depending on the ways the mappings  $f$ ,  $s$  and the predicate  $p$  are spelled out.

## A.2 Documents and Annotations

The theory outlined in the previous subsection addresses the issues of path expression, type construction and subtype queries. It does not address the issue of document reference, which is addressed in this subsection. We introduce a mapping from some types to pairs of types and attribute names:

$$\text{doc} : A \rightarrow D, d \quad (24)$$

where  $A, D$  are types,  $d$  is an attribute name of  $A$ , and  $A.d = D$ . Intuitively,  $D$  is considered a document type.  $A$  is considered an annotation type, obtained by performing annotation on a text attribute of the document.  $d$  is the reference from the annotation back to the original document. Exactly which attribute  $t$  of  $D$  is regarded as text is of little concern to us here. Note that the annotations produced by the same TAE but on different text field of the same document type  $D$  would be considered as different types.

Annotation types are the primary mechanisms for combining structured and unstructured data. For example, given set of documents  $D$ , we might perform multiple annotations  $A_1, A_2, \dots$  on  $D$ , resulting in

$$\text{doc}(A_1) = (D, d_1), \quad (25)$$

$$\text{doc}(A_2) = (D, d_2), \quad (26)$$

$$\dots \quad (27)$$

A query might ask for all the annotations  $a_1 \in \mathbf{objs}(A_1)$ ,  $a_2 \in \mathbf{objs}(A_2)$ ,  $\dots$ , such that  $a_1.d_1 = a_2.d_2, \dots$ , in addition to whatever predicates these objects must satisfy. Such queries where all the annotations are joined by common documents occur quite often in practice. As a convenience to the user, we define a new operator  $\mathbf{A}$ , which works in the following way: Given a mapping  $f$  from some names to annotation types

$$f : x_i \rightarrow A_i, \quad \text{where } \text{doc}(A_i) = D_i, d_i, \quad (28)$$

a new annotation type  $A = \mathbf{A} f$  is defined similar to  $T f$ , except that it has an additional attribute  $d$  such that  $A.d = D_i$  for all  $i$ . Therefore it is only defined when all the annotations  $A_i$  share the same document type, which also becomes its own document type. Furthermore, for any object  $a \in \mathbf{objs}(A)$ , the attribute  $a.d = a.x_i.d_i$  for all  $i$ . That is, an object  $a$  in  $A$  is made up of objects  $a_i$  in  $A_i$ , such that all the  $a_i$  are annotations on the same document, plus an attribute  $a.d$  that equals the document.

Clearly, the type  $A$  thus defined is also an annotation type, with  $\text{doc}(A) = (A.d, d)$ . Similar to the operator  $T$ , we can use the operator  $A$  to form general queries of the form  $\pi_s \sigma_p A f$ .

### A.3 Simple relational backend

The object model as described above is independent of actual storage schemes. Due to its similarity to relational models, it is relatively easy to translate object model concepts into a relational models for storage. We describe here the most straight forward translation. Other translations are possible, having different performances and optimization issues. It is also possible to translate the object model into XML data models. When such backend storage becomes practically available, we intend to make corresponding translations available as well. This will not be discussed any further here.

In the simple backend scheme, the object model is translated into relational model according to the following rules:

- A type  $A$  is translated into a table  $t = \text{tab}(A)$ .
- An attribute  $x$  of  $A$  is translated into a column  $c = \text{col}(A, x)$ . An additional column  $i = \text{id}(A)$  acts as primary key of the table  $t$ .
- An object  $a$  of type  $A$  is translated into a row in  $\text{tab}(A)$ .
- If  $x$  is an atomic attribute, then the value  $a.x$  is stored in the column  $c$  of table  $t$ .
- If  $x$  is a non-atomic attribute, then the value  $a.x$  is stored in the foreign table  $t_1 = \text{tab}(A.x)$  with id column  $i_1 = \text{col}(A.x)$ . The column  $c$  in table  $t$  is a foreign key reference to  $t_1, c_1$ .

With these rules, one additional type introduced into the object model corresponds to one or several additional tables in the backend relational model. One additional object corresponds to one extra row in the tables corresponding to the object type and the non-atomic types of all its paths. Atomic attributes are stored in place in the tables while non-atomic types are stored in foreign tables.

This translation scheme, together with the type aspects of the object model, such as subtype relations and schemas, can themselves be described in terms of relational tables, called the metadata tables. A query to the object model can be uniquely translated in to an SQL query to the backend relational model according to information in the metadata tables. Consider a query in the form

$$\pi_s \sigma_p T f, \quad (29)$$

where  $s$  is a map from some names to paths,  $p$  is a predicate, and  $f$  is a mapping from some names to types. The translation involves the following the main steps

- Resolving path expressions appearing in either  $s$  or  $p$ . A path of the form  $x_1.x_2 \dots$  introduces additional table aliases for each link.

- Constructing join conditions associated with path expressions. These join conditions are produced by the foreign key references related to each link in the path.
- Construcing join conditions associated document reference. If the query is based on operator  $A$  instead of  $T$ , additional join conditions for the common document references are introduced.

## B Probabilistic OLAP

In this section we outline the mechanics of our solution for Case 3 from Section 6 Let  $A_1, A_2, \dots, A_k$  denote the attributes over all the dimensions. The data is given as a table of records where each record contains an assignment of values to  $A_1, A_2, \dots, A_k$ , and a probability distribution of a single uncertain measure, called *opinion*<sup>4</sup> denoted by  $O$ . This table is constructed by obtaining  $A_1, A_2, \dots, A_k$  from  $D_s$  and  $O$  from  $D_e$  (cf. Figure 2).

The statistical model is formalized by considering the joint probability distribution  $P(O, A_1, A_2, \dots, A_k)$ , which factors into the product of  $P(O|A_1, \dots, A_k)$  and  $P(A_1, \dots, A_k)$ . The first term  $P(O|A_1, \dots, A_k)$  models the uncertainty in the opinion with respect to the attribute assignment. The other term in the product can be viewed as the *weight* associated with that opinion. The joint distribution  $P(O, A_1, \dots, A_k)$  is obtained by optimizing a KL-divergence objective function as shown in [20, 8]. In this setting, each record in the data is interpreted as an empirical conditional probability distribution on the opinion. Specifically, let  $a_1, a_2, \dots, a_k$  denote the assignment of values to  $A_1, A_2, \dots, A_k$ , respectively. Then, the probability associated with opinion value  $o$  is denoted by  $\hat{p}(O = o|A_1 = a_1, \dots, A_k = a_k)$ , or simply,  $\hat{p}(o|a_1, \dots, a_k)$ .

The approach is illustrated using our running example where *Topic = Brake* is the measure. The associated attributes are MODEL, CATEGORY, STATE and REGION. The attributes can be grouped into two hierarchies: MODEL determines CATEGORY and STATE determines REGION. Consider the query “What is the chance that brake problems occur in New York for Chevy?”. This corresponds to the query  $P(\text{BRAKE}|\text{STATE} = \text{'NY'}, \text{CATEGORY} = \text{'Chevy'})$  on the statistical model. Note that this is equivalent to  $P(\text{BRAKE}|\text{STATE} = \text{'NY'}, \text{REGION} = \text{'East'}, \text{CATEGORY} = \text{'Chevy'})$  by the hierarchical constraint. Now consider the query “What is the probability of brake related problems according to the category of the automobile?” This is equivalent to a set of queries,  $P(\text{BRAKE}|\text{CATEGORY})$  one for each possible category. The answer is a set of two distributions, one for trucks and another for sedans.

Queries on the the statistical model can be answered from the joint distribution  $P(O, A_1, \dots, A_k)$ . Let us consider a query on the statistical model  $P(O|\mathbf{a})$ , where  $\mathbf{a}$  is

<sup>4</sup>The term opinion originates from the fact that the approach in Pro-OLAP has been motivated by *opinion pooling*, a well-known statistical method for obtaining consensus distributions.

an assignment to the attributes  $A_1, A_2, \dots, A_k$ . The assignment  $\mathbf{a}$  could either be a complete or a partial assignment depending on the level of the query. Now, for any  $o$ ,

$$\begin{aligned} P(o|\mathbf{a}) &= \frac{P(o, \mathbf{a})}{P(\mathbf{a})} \\ &= \frac{\sum_{\mathbf{a}'} P(o, \mathbf{a}, \mathbf{a}')}{\sum_{\mathbf{a}'} P(\mathbf{a}, \mathbf{a}')} \end{aligned} \quad (30)$$

$$= \frac{\sum_{\mathbf{a}'} P(o, \mathbf{a}')}{\sum_{\mathbf{a}'} P(\mathbf{a}')} \quad (31)$$

$$= \frac{\sum_{\mathbf{a}'} P(o, \mathbf{a})}{\sum_{o'} \sum_{\mathbf{a}'} P(o', \mathbf{a}')} \quad (32)$$

In the above equations,  $\mathbf{a}'$  ranges over all complete assignments consistent with  $\mathbf{a}$  and  $o'$  ranges over all possible measure values. The derivation from Equation 30 to Equation 31 follows from the fact that  $\mathbf{a}'$  determines  $\mathbf{a}$ , so  $P(\mathbf{a}, \mathbf{a}') = P(\mathbf{a}')$ . All terms on the right hand side are known from the joint distribution, so  $P(o|\mathbf{a})$  can be computed. In essence, we are marginalizing over all complete assignments by assigning the missing attributes to all possible values. A similar equation can be derived to answer range queries where the query explicitly specifies a range along each dimension to be aggregated.

### Mapping to SQL

We use a star schema to store the probability distribution  $P(O, A_1, A_2, \dots, A_k)$ . This enables us to compute the aggregates as described in Equation 32 as a star join query. A star schema consists of a fact table and set of dimension tables. In our case, the fact table has  $n + m$  attributes, where  $n$  denotes the number of dimensions and  $m$  denotes the number of possible values for the opinion measure. The  $n$  dimension columns correspond to the leaf level attributes of the dimensions. Each leaf attribute completely determines the other attributes in the corresponding hierarchy. The  $m$  measure columns store the probabilities corresponding to each opinion value. Thus, the measure column corresponding to the opinion value  $o$  stores  $P(o, a_1, a_2, \dots, a_k)$  for each row that has the leaf attributes among  $a_i$  in the corresponding dimension column. There are  $n$  dimension tables, with each table having the attributes corresponding to that dimension. Tuples in the dimension table encode the hierarchy in that dimension. Each dimension table joins with the fact table through the leaf level attribute of that dimension. For example, Figure 10 shows the star schema for *Brake* data. Since the measure can take two values, there are two corresponding columns *BRAKE\_YES* and *BRAKE\_NO* to store the probabilities.

Now consider Equation 32 used to compute a point query. It can be seen that the set  $\mathbf{a}'$  of tuples consistent with  $\mathbf{a}$  is in fact equivalent to the star join of the fact table with the dimension tables with constraints on attributes of the dimension tables set to values corresponding to those attributes that are assigned in  $\mathbf{a}$ . This implies that the numerator of Equation 32 can be obtained by a simple *Sum*

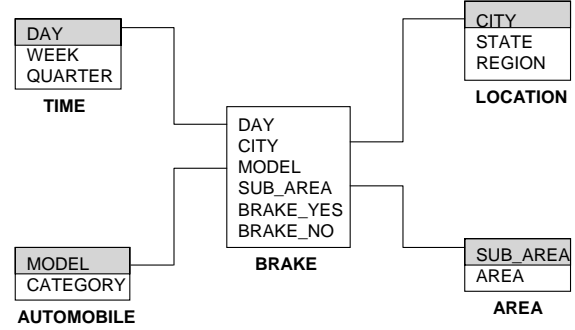


Figure 10: Star Schema

aggregation on the star join query. Since  $o'$  ranges over all opinion values, the denominator will have the *Sum* aggregate over all the measure columns. With this mapping, Equation 32 can be written as a SQL query over the star schema. The query can be optimized to eliminate dimension tables that don't contribute any constraints from the star join. We will elaborate with a few queries on the example star schema.

**Query 10.** *What is the probability of brake related problems in New York for Chevs?*

*This query is equivalent to:*  
 $P(\text{BRAKE} = \text{'YES'} | \text{STATE} = \text{'NY'}, \text{CATEGORY} = \text{'Chevy'})^5$

```
SELECT SUM (BRAKE_YES) / ( SUM (BRAKE_YES)
    + SUM (BRAKE_NO))
FROM BRAKE, LOCATION, AUTOMOBILE
WHERE LOCATION.CITY = BRAKE.CITY AND
AUTOMOBILE.MODEL = BRAKE.MODEL AND
LOCATION.STATE = 'NY' AND
AUTOMOBILE.CATEGORY = 'Chevy'
```

*Note that in this case, dimension tables TIME and AREA have been dropped from the join as there is no constraint on their attributes*

**Query 11.** *What is the probability of brake related problems according to the Category of the automobile? This is equivalent to a set of point queries  $P(\text{BRAKE} = \text{'YES'} | \text{CATEGORY})$ , one for each possible Category.*

```
SELECT CATEGORY,
    SUM (BRAKE_YES) / ( SUM (BRAKE_YES)
    + SUM (BRAKE_NO))
FROM BRAKE, AUTOMOBILE
WHERE AUTOMOBILE.MODEL = BRAKE.MODEL AND
GROUP BY CATEGORY
```

*In this case, a group by clause is added to groups tuples corresponding to each make.*

<sup>5</sup>We have used  $P(\text{BRAKE} = \text{YES} | \text{City} = \text{NY}, \text{Category} = \text{Chevy})$  instead of  $P(\text{TOPIC} = \text{Brake} | \text{City} = \text{NY}, \text{Category} = \text{Chevy})$  for readability.

As shown in these examples, the opinion aggregation operators can be mapped to existing SQL operators without the need for any defining new operators. This is quite powerful since it makes it possible to use existing OLAP infrastructure and any existing OLAP optimizations like pre-computation or specialized query evaluation algorithms.