Toward Automated Large-Scale Information Integration and Discovery

Paul Brown, Peter Haas, Jussi Myllymaki,
Hamid Pirahesh, Berthold Reinwald, Yannis Sismanis

IBM Almaden Research Center
{pbrown1, phaas, jussi, pirahesh, reinwald, syannis}@us.ibm.com

Abstract. The high cost of data consolidation is the key market inhibitor to the adoption of traditional information integration and data warehousing solutions. In this paper, we outline a next-generation integrated database management system that takes traditional information integration, content management, and data warehouse techniques to the next level: the system will be able to integrate a very large number of information sources and automatically construct a global business view in terms of "Universal Business Objects". We describe techniques for discovering, unifying, and aggregating data from a large number of disparate data sources. Enabling technologies for our solution are XML, web services, caching, messaging, and portals for real-time dashboarding and reporting.

1 Introduction

Efficient business management requires a flexible, unified view of all data in the enterprise. Many mid-sized and large companies are now facing the challenging problem of bringing together all of their data in a unified form. Enterprise business data is typically spread across hundreds or thousands of sources: inside applications such as SAP, Siebel, and PeopleSoft, on web sites, as syndicated data feeds, in content-management warehouses and marts, in email archives, in spreadsheets and other office documents, and inside a tremendous variety of custom applications. Data collected over a long period of time is juxtaposed with ready-made data sources that come as part of turnkey systems.

Disparate data sources are brought together through business acquisitions and mergers. The various operational systems within an enterprise are usually isolated, have data that is inconsistent both in format and semantics with other operational systems, and offer different end-user tools for data retrieval and reporting. There is a marked lack of data integrity; for example, a given entity can have different names in different operational systems, distinct entities can have the same name, and an entity may exist in some systems but not in others. Adding to the problem, most traditional databases were

T. Härder and W. Lehner (Eds.): Data Management (Wedekind Festschrift), LNCS 3551, pp. 161-180, 2005.

not designed with future data integration in mind. These databases, which place a premium on data integrity and consistency, are usually encased in a very large body of difficult-to-change procedural code that is inextricably entwined with the schema semantics.

The major costs of an integration project are usually incurred while trying to "understand the data," i.e., trying to figure out the schema of each input data source, the constraint rules that govern the data in each schema, and, perhaps most importantly, the rules that describe the relationships between data from different sources. These assertions are borne out in the traditional setting for data integration, namely data warehousing. Traditional warehouses try to provide a unified view of the data while hiding the diversity of the underlying operational systems. Experience with data-warehousing projects suggests that the capital cost of integration is very high, and an average design time of six months to two years causes many projects to fail. Manual schema integration constitutes the bulk of the expense. At least one vendor [11] estimates that labor costs for large warehousing projects currently comprise 70% of the total costs, and we believe that the relative labor costs for traditional warehousing solutions will only increase in the future. The initial per-document cost of enterprise content management systems is less than that of data warehouses, but the deep document-analytic functionality provided by content management systems is frequently inferior to the BI capabilities of data warehouses.

In this paper, we outline a next-generation integrated database management system (DBMS) that brings together information integration, content management, and data warehousing with business intelligence. The integrated DBMS (1) taps into the data flow between operational systems and captures raw data, (2) automatically constructs a global business view in terms of Universal Business Objects (UBOs), (3) provides warehousing functionality for the UBOs such as cube queries, advanced analytics, and efficient propagation of data changes, and (4) provides rich interfaces for browsing and searching UBOs that hide the details and variety of the underlying operational systems. We take a data-centric integration approach in that we capture raw data only, and do not rely on any schema information. To ensure that the system scales to very large numbers of information sources, we employ asynchronous messaging and crawling techniques similar to those found in web-scale applications such as Google. We apply machine-learning techniques to map information across disparate sources, and develop new algorithms to map information sources into higher-level business artifacts; these artifacts can either be discovered by the system or introduced into the system through enterprise data dictionaries, taxonomies, and ontologies. We introduce a new query paradigm that takes queries over UBOs and converts them into queries over specific information sources based on discovered relationships between their sources. The enabling core technologies for this next-generation integrated DBMS are XML, web services, tools for information dissemination and dashboards, frameworks for unstructured information management (such as IBM's UIMA architecture) and advanced search capabilities.

2 A Next-Generation Integrated Database Management System

In this section, we outline the architecture of a next-generation integrated DBMS, giving a high-level view of the components and the flow of data through the system. We also describe a hypothetical end-to-end application scenario.

2.1 Architecture

Fig. 1 outlines the architecture of the integrated DBMS. The architecture comprises three layers: the operational systems layer, the integrated DBMS layer, and the business applications layer. The operational layer at the bottom of the figure displays some typical data sources that might exist in an enterprise, such as CRM and HR applications with proprietary data formats or database schemas, content management systems, and Office productivity tools. The operational systems in an enterprise interoperate today through synchronous or asynchronous communication such as message queues, email, paper, and so forth, typically in a point-to-point and ad hoc fashion. In the future, web services and the "enterprise service bus" (ESB)—which provides mediation and transformation services—will play an important role in facilitating interoperability. The communication trail for a business transaction can usually be reconstructed based on application-specific information in the exchanged data. For example, we observe that a customer sent an order referencing the customer ID, that the seller sent an invoice referencing the order ID, and so forth. Whereas this kind of information is sufficient for reasoning about point to point operational interactions, approaching an enterprise top-down from the business layer requires an integrated business view to answer high-level queries or automatically produce high-level reports. The goal of our integrated DBMS is to close the gap between the business layer and the operational layer.

The integrated DBMS taps into the operational data flow to capture input data. The system analyzes the input data, assigning the incoming data values to disjoint "data sets" and then computing a synopsis for each data set. The synopsis is then used to compute one or more "signatures" for each data set. The signatures can then be used to derive similarity measures between a pair of data sets. In particular, by identifying similarities between an input data set and a reference data set, the system can identify low-level business artifacts such as "telephone number" or "North American surname." Using methods from graph theory, these low-level identifications can be combined and used to identify higher-level artifacts such as "customer", "order", "account", and so forth. We call these high-level artifacts *Universal Business Objects* (UBOs). The data sets that define a UBO may come from different data sources. Relationships between the UBOs can also be identified. Enterprise data dictionaries, ontologies, and reference data all may be incorporated in the foregoing process. The system permits the UBOs and other discovered metadata to be cached or instantiated in the integrated DBMS for purposes of indexing, querying, etc., just like ordinary warehouse data today. The

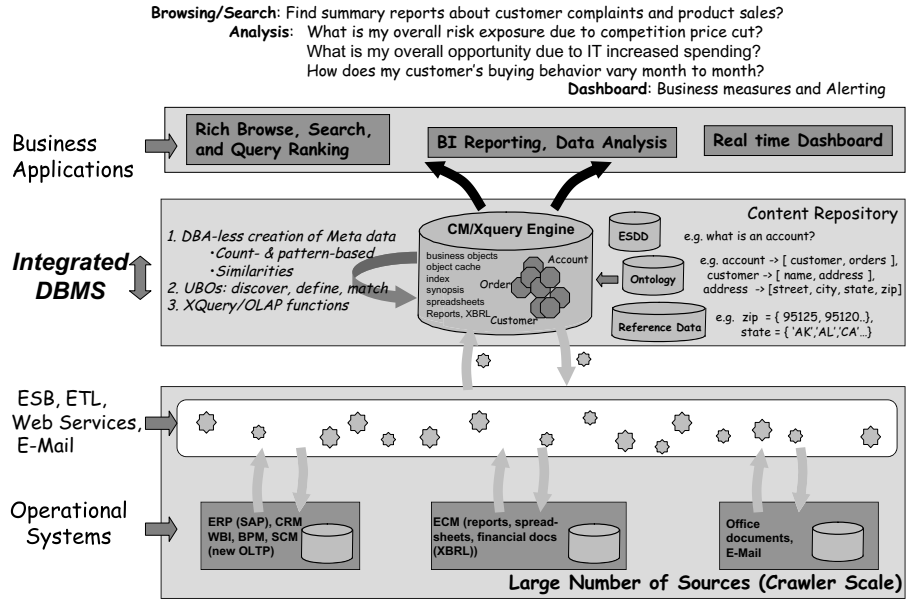

Fig. 1 System Architecture

UBOs are also exposed to the business applications for rich browsing, search, report generation, data analysis, UBO updating, and real-time dashboards.

2.2 Repository Dataflow

In this section, we describe in more detail the dataflow within the integrated DBMS (Fig. 2). The details of the various components in the dataflow are described in subsequent sections.

The integrated DBMS takes in a large collection of XML documents. The documents in the collection are partitioned into mutually disjoint sets, and we say informally that documents in the same partition come from the same *data source*. Heuristically, documents from the same data source are similar to one another in origin, destination, structure, and content. For example, a given data source might comprise the XML messages returned from a specified web service in response to a particular kind of message, or perhaps the source might comprise purchase orders generated by a specified application.

The data values in the XML documents that come from a given data source are assigned to disjoint *data sets*, as follows. Each XML document from a given data source can be viewed as a "tree data structure", with data values stored at the leaf nodes. Each leaf node (as well as each internal node) of the complex structure is uniquely identified by an XPath expression; the set of XPaths (i.e., nodes) for a given data source can be computed simply by submitting every XML document from the data source to a generic SAX parser. There is a data set associated with each leaf node, and for the *i*th leaf node

we denote the corresponding data set as $ds_i = \{v_1, v_2, ..., v_n\}$, where v_j is the data value at the ith leaf node in the jth XML document from the specified data source. Observe that ds_i is a multiset, because the same data value can appear in multiple documents. Also observe that each data set can be identified uniquely by data source and XPath expression for the corresponding leaf node. The *domain* associated with a data set *ds* is the set of distinct values obtained from *ds* after removal of duplicates.

As described in subsequent sections, the DBMS computes a synopsis of each data set from a given data source. The synopsis contains statistical information about the data set such as sample values and histogram data. The resulting synopses for the data source are stored in the *DataSet* repository. For each data set, the DBMS uses the synopsis to derive one or more *signatures*; a signature summarizes a specific characteristic of the data set that is pertinent to evaluating the similarity of the data set to other data sets. The signatures are stored with the corresponding data set in the *DataSet* repository. In our initial prototype, the signatures comprise a space-constrained concise Bernoulli sample and a hash-counting data structure. Patterns are, for example, rules for phone numbers, credit card numbers, zip codes, and so forth. The pattern signature is derived from the synopsis using an automaton-based technique that is described in a subsequent section.

Next, the DBMS uses the signatures to compute similarities between data sets. The computed similarities for each related pair of data sets are combined via a *Synthesize* function and stored in the *Similarities* repository. When there are N input data sets, a direct pairwise comparison of all these data sets is an $O(N^2)$ operation. Since N is usually very large, such a computation is typically too expensive. The DBMS therefore first compares each input data set to one or more *reference* data sets, where a reference data set is typically derived from a data dictionary, ontology, or from a previous discovery phase of the DBMS. Such a comparison is typically an $O(NM)$ operation, where M is the number of reference data sets. Because M is typically smaller than N by orders of magnitude, this computation is usually feasible. Then, when comparing input data sets, only data sets that are both similar to a common reference data set need to be compared.

The similarity-identification process now proceeds from low-level data sets to higher-level structures. After computing similarities for leaf-level nodes in the XPath only (e.g. city, zip code, street name), the DBMS invokes the Matcher component to compute similarity metrics between internal nodes, e.g., address, customer. The Matcher algorithms are based on graph theory. In addition to computing similarity metrics, the DBMS also identifies constraints between pairs of data sets such as key-to-foreign-key relationships and functional dependencies; the data sets in a pair may be from the same or different data sources. Finally, given all the similarities between data sets, the similarities between internal nodes, and the discovered constraints, the DBMS applies a recursive splitting algorithm to discover UBOs.

2.3 A Hypothetical Application Scenario

To make the forgoing discussion more concrete, we now describe a hypothetical (small scale) end-to-end application scenario. This scenario encompasses a collection of data

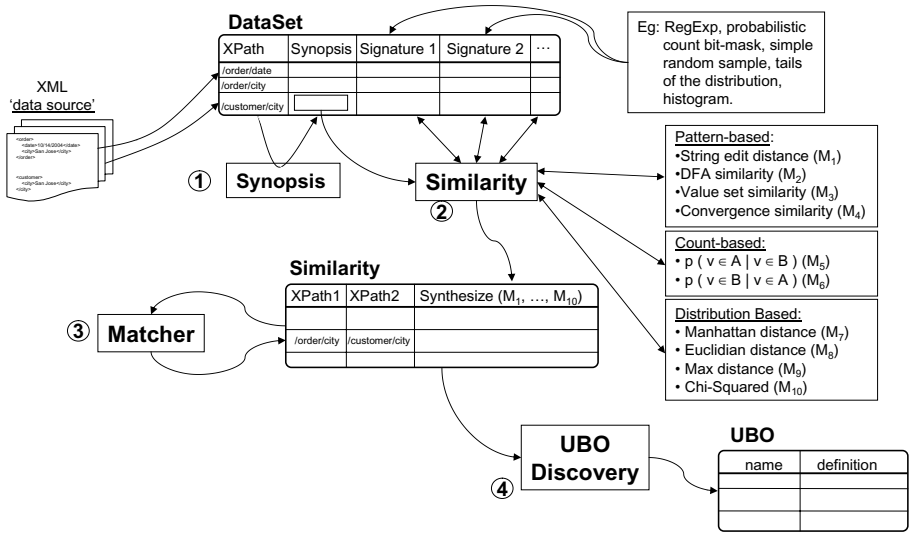

Fig. 2 Repository Dataflow

sources at the lowest level and a client spreadsheet application at the highest level (see Fig. 3). We capture XML documents from four data sources. The system analyzes the data sources, discovers the data sets, and computes both XPath identifiers and synopses. Suppose that three sources of reference data are available: product category information from the United Nations Standard Products and Services (UNSPSC), time information (i.e., a calendar), and location information from the United States Census Bureau. The system creates synopses for the reference data as well.

The system would then analyze the data sources and reference data to compute similarities between data sets and composite nodes. Ultimately the system would identify three new UBOs W', X', and Y' along with their domains, constraints, and relationships both to each other and to the metadata. Semantic analysis of W', X', and Y' might suggest names such as Customer, Product, and Order. Applying some OLAP tools, the system can then generate a spreadsheet containing the quarterly sales figures.

3 Automated Similarity Analysis

In this section, we describe in more detail the various techniques that the DBMS uses to create synopses, compute signatures, derive similarity measures for both data sets and internal XML nodes, and discover constraints.

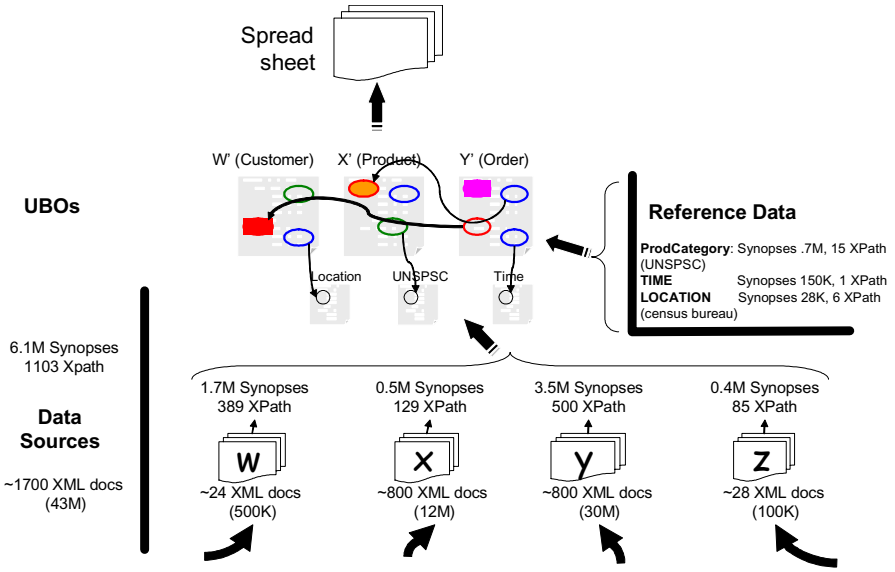

Fig. 3 Example

3.1 Synopsis Creation

The synopsis maintained for a data set in our current prototype comprises several components. The first component is a space-constrained sample from which a uniform random sample can be extracted as needed. If the number of distinct values in the incoming data is small enough, the sample can be stored in a concise form similar to that described in [5].

The second key component of the synopsis is a hash-counting data structure. The idea is that each incoming data element is fed into a "probabilistic counting" algorithm; see, e.g., [2]. Such algorithms estimate the number of distinct values in a data set in a single pass using a fixed amount of memory. The value of the incoming element is hashed, and this hash value is used to update an appropriate data structure. The data structure is then used to estimate the number of distinct values. In the most naïve algorithm, the data structure is simply a bit vector, where each bit corresponds to a hash bucket. For most algorithms, it is the case that two hash-data structures created from two disjoint data sets can be combined into a single structure of the same type and used to estimate the number of distinct values in the union of the data sets.

Other information maintained in the synopsis includes the data set cardinality, the path expression that specifies the data set, and the k highest and k lowest data values encountered, along with corresponding frequency counts.

3.2 Signature Creation

As mentioned previously, one or more signatures are computed for each data set, and are subsequently used to derive similarity measures. The signatures currently used in our prototype are the synopsis hash-counting structure itself and a regular-expression representation of a pattern. We discuss statistics-based signatures, as well as pattern-based signatures and hybrids of the two signature types.

Statistics-Based Signatures. A classical statistics-based signature is a lossy histogram representation of the data value distribution in the data set. Here, the set of possible data values is partitioned into disjoint buckets and a count is maintained for each bucket. More sophisticated histograms are available; see [7] for a comprehensive taxonomy of histogram types. We are currently investigating other kinds of statistics-based signatures. We focus here on pattern-based signatures, because such signatures are relatively novel. Also, as discussed below, it appears possible to develop signatures that are both pattern- and statistics-based.

Pattern-Based Signatures. Pattern-based signatures complement statistics-based signatures and provide an alternate way to characterize a data set. Pattern-based signatures can reveal similarities between data sets that statistics-based signatures will miss. For example, one data set might contain telephone numbers in the 408 area code while another may contain telephone numbers in the 650 area code. A statistics-based synopsis can be computed for each data set, but comparing the two data sets via their statistics-based synopses might reveal no similarity between the data sets because they have no values in common. Yet clearly the two sets are similar because both contain telephone numbers. A similar problem occurs when the values of two data sets are formatted differently. For instance, the data sets might contain the same set of telephone numbers but use different delimiters to separate an area code from the local phone number.

Pattern-based signatures are computed by taking the sample data values stored in the synopsis of a data set and determining a regular expression that is matched by (almost all of) the values in the data set. Pattern discovery is based on the theory of deterministic finite automata and builds on the existing work on induction of regular languages and DFAs [1, 3, 8]. A matching automaton consists of nodes, transitions and symbol ranges and is constructed incrementally for a given input value set. A node matches a symbol (character) or range of symbols appearing in a specific position of an input value. A directed arc between two nodes-source and destination-means that the source symbol immediately precedes the target symbol. Fig. 4 shows a sample automaton that matches postal codes defined by “950xx” where x can be any digit.

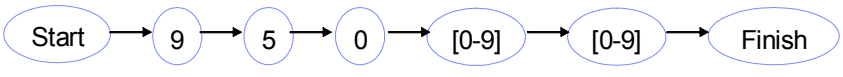

Fig. 4 Pattern for Matching Postal Codes 950xx

The pattern-signature computation includes a set of expansion steps on the automaton to make it more general. For instance, if the input value set consists of digits 0-3, 5, and 7-9, we might want to expand the set to cover all digits 0 through 9. We measure expansion as the ratio of the size of the expanded set to the size of the input set, so for this example the expansion ratio would be 10 to 8, or an expansion factor of 1.25. Expansion reduces the accuracy of the pattern, as in this case digits 4 and 6 were not in the input set. A user-defined control parameter sets the upper limit for the maximum allowed expansion factor.

A set of reduction steps are performed to simplify the pattern. Values that have common prefixes share nodes in the automaton, and nodes which match identical or similar symbol sets (after potential expansion) and share predecessor and successor nodes can be combined. Fig. 5 shows how postal codes 950xx and 951xx are combined into a single, compact pattern. Again, loss of accuracy is possible and a control parameter ensures that the pattern is not overly generic.

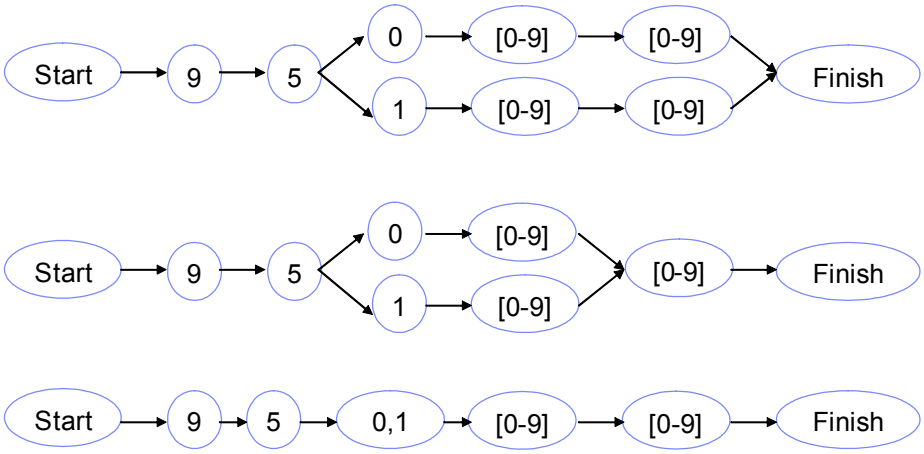

Fig. 5 Merging of Nodes in Pattern

Repeating symbol sequences are identified and the minimum and maximum repetition length is recorded in the automaton. This allows the DBMS to detect patterns such as the one shown in Fig. 6—a sequence of digits of length one or two.

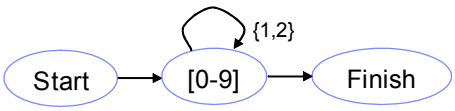

Fig. 6 Repeating Pattern

To facilitate more accurate comparison and matching, patterns may be subjected to further simplification by removing delimiters. A delimiter is defined as a node in the automaton that matches exactly one symbol and is visited by all paths through the au-

tomaton. A delimiter may appear anywhere in the pattern and is easily identified and removed from the automaton.

An alternative way to use delimiter information is to break compound values into their constituent parts. For instance, a telephone number can be broken into three parts (area code, exchange number, and local extension) and signatures can be computed for each part. This enables the DBMS to match the area code part of a telephone number with another data set that contains area codes only. Observe that the signatures for the different parts can be statistics-based; in this manner, we can develop signatures that are a hybrid of the statistics-based and pattern-based approaches.

Our current prototype externalizes discovered pattern signatures as regular-expression strings because of their compact form and convenience for storage and display purposes. We also note that XML Schemas and XML parsers have built-in support for regular expressions and regular expression engines are readily available. Table 1 shows several regular expressions discovered for business data sets.

Tab. 1 Sample Regular Expressions for Business data Sets

XPath	Pattern
/SAP4_CustomerMaster/State	A[LR] | C [AO] | [FI] L | [HW] I | [MV] A |
/SAP4_CustomerMaster/PostalCode1	N[CV] | OH | TX
/SAP4_CustomerMaster/PhoneNumber1	950 [0-9] [0-9]
/SAP4_CustomerMaster/NumberOfEmployees	[0-9] [0-9] [0-9] \ – [0-9] [0-9] [0-9] \ - [0-9] [0-9] [0-9] [0-9]
/SAP4_CustomerMaster/. . . /OrgEntity1	[0-9] {2,3}
/SAP4_CustomerMaster/. . . /OrderProbability	US[0-9] [0-9]1?
/SAP4_CustomerMaster/. . . /NumDeliveryAllowed	[0-9] [0-9] [123]
/SAP4_Order/OrderId	[0-9] {1,4}
/SAP4_Order/. . . / SAP4_OrderSoldToInfo:PostalCode1	950[0-9] [0-9] \ – [0-9] | [0-9] {1,2}
/Siebel_BCAccount/RevenueGrowth	

3.3 Similarity Computation Between Data Sets

After computing one or more signatures for the data sets, the DBMS then computes similarity measures between pairs of data sets; such a measure equals 0 if there is no apparent similarity between a pair of data sets and becomes larger as the degree of similarity becomes greater. Either both data sets are input data sets or one data set is an input data set and the other data set is a reference data set.

We assume that for the *i*th signature type there exists a similarity metric m_i that measures the similarity between any two signatures of that type. Then, given two data sets ds_1 and ds_2, along with signatures $s_1, s_2, ..., s_k$, the most general form of a similarity measure is

$$\text{Similarity}(ds_1,ds_2) = \text{Synthesize}[m_1(s_1(ds_1),s_1(ds_2)), \ldots, m_k(s_k(ds_1),s_k(ds_2))],$$

where the function *Synthesize* combines the multiple similarity measures into a single measure. (Actually the system allows multiple similarity measures to be computed for each signature, as in Fig. 2). We are currently investigating possible *Synthesize* functions.

Recall that the domain of a data set is the set of distinct values that appear in the data set. Often the domain of a reference data set coincides with the reference data set itself, and we call such a reference data set a *domain set*, which we abbreviate simply as a *domain* when there is no risk of confusion, i.e., the reference data set is a true set rather than a multiset. If there is a strong similarity between an input data set and a domain set, we say that we have "discovered the domain" of the input data set. Similarly, if two input data sets are each similar to the same domain set, then we say that the two input data sets "share the same domain." If the system is computing a similarity measure between an input data set and a domain set, then the system measures the similarity between the domain of the input data set and the domain set.

In the rest of this section, we describe similarity measures m_i that can be computed for different kinds of signatures, with an emphasis on the kinds of signatures that the prototype currently computes. These measures can be used separately or combined together.

Similarity for Hash-Count Signatures: As discussed previously, one signature that the DBMS computes is the hash-counting structure created during execution of a probabilistic counting algorithm. This signature is identical for any two data sets having the same domain. Thus a similarity measure on hash-count signatures is ideal for comparing an input data set with a domain set, since the domain of the input data set does not need to be computed explicitly. One such measure is the Jaccard metric, which is a normalized measure of the intersection of the domains of two data sets. The idea is to use the probabilistic counting algorithm associated with the hash count signature to estimate C1 = $Card[Domain(ds_1)]$ and C2 = $Card[Domain(ds_2)]$. Then the two hash-count structures are merged into a combined hash-count structure and the probabilistic counting algorithm is then applied to obtain an estimate of

$$C3 = \text{Card}[\text{Domain}(ds_1) \cup \text{Domain}(ds_2)].$$

Then the size of the intersection of the two domains is computed as C4 = C1 + C2 - C3, and the Jaccard metric is computed as J = C4 / C3.

Similarity for Histogram Signatures. A histogram summarizes the distribution of data values within a data set. Suppose for simplicity that histogram signatures of data sets ds_1 and ds_2 are exact and can be represented as vectors $f_1 = (f_{1,1},...,f_{1,k})$ and $f_2 = (f_{2,1},...,f_{2,k})$, where $f_{i,j}$ is the relative frequency of the *j*th value in the *i*th data set. Then the similarity between the two signatures can be computed, for example, as $1/d(f_1,f_2)$, where *d* is any of the standard vector space metrics, e.g., Euclidean distance $d(f_1,f_2) =$

$((f_{1,1}-f_{2,1})^2 + ... + (f_{1,k}-f_{2,k})^2)^{1/2}$, L_1 distance $d(f_1,f_2) = |f_{1,1}-f_{2,1}| + ... + |f_{1,k}-f_{2,k}|$, L_∞ distance $d(f_1,f_2) = \max_j |f_{1,j}-f_{2,j}|$ or the symmetric Kullback-Liebler distance $d(f_1,f_2) = (f_{1,1}log(f_{1,1})/log(f_{2,1})) + \ldots + (f_{1,k}log(f_{1,k})/log(f_{2,k})) + (f_{2,1}log(f_{2,1})/log(f_{1,1})) + \ldots + (f_{2,k}log(f_{2,k})/log(f_{1,k}))$. Variants of these metrics are available when the histogram is lossy or the data values are real numbers.

Similarity for Pattern Signatures. We have devised several similarity metrics for pattern signatures. One metric, which is naive but easy to compute and useful, is obtained by simply comparing the regular expression strings of each pattern. For example, we can first compute the Levenshtein string-edit distance [8] between the regular-expression strings, divide it by the length of the longer string, and then convert the normalized distance d to a similarity measure by setting m = 1 - d. For instance, patterns "95xxx" and "96xxx" look very similar even though the underlying domains are different. On the other hand, two patterns may define the same domain in a different way, resulting in an incorrect assessment of their similarity. For instance, patterns "[1-3]" and "1 | 2 | 3" look very different but define the same domain.

A value-based similarity measure is slightly more expensive but more reliable. For this measure, we count the number of elements in one data set that satisfy the pattern for the other data set, and vice versa. Data values may be generated from the pattern or retrieved from the sample that is stored in the synopsis. Consider the following two patterns "200[0-9]" and "20[01][0-9]". The first pattern defines the domain "first decade of 2000" and the second "first two decades of 2000". All ten distinct values generated from the first pattern (2000, 2001, …, 2009) are accepted by the second pattern. Half the numbers generated from the second pattern are accepted by the first pattern. The sum (10 plus 10) is divided by the total vocabulary size (10 plus 20) and we get a similarity score of 2/3. Note that for the patterns of the earlier example ("[1-3]" and "1 | 2 | 3") the similarity score would be a perfect 1.0.

3.4 Similarity Measures Between Complex Structures

We now discuss how the DBMS defines similarity measures between pairs of complex structures. The goal is to identify complex structures that are structurally similar to other complex structures; we often call this process "schema matching". The inputs to this process are structure definitions and leaf-node to leaf-node similarities. Structure definitions, or parent-child relationships, can be extracted from the synopses of each data source.

Fig. 7 illustrates a scenario where SRC A, SRC B and SRC C represent the structures of three data sources that all contain the same information, albeit structured differently. They describe a number of products, sales of those products, and product deliveries consequent to sales. In SRC A, product information contains sales information, which contains delivery information. This arrangement might be common in a product support department where information is organized by product. SRC B might correspond to a shipping department. Information is organized by delivery, with associated

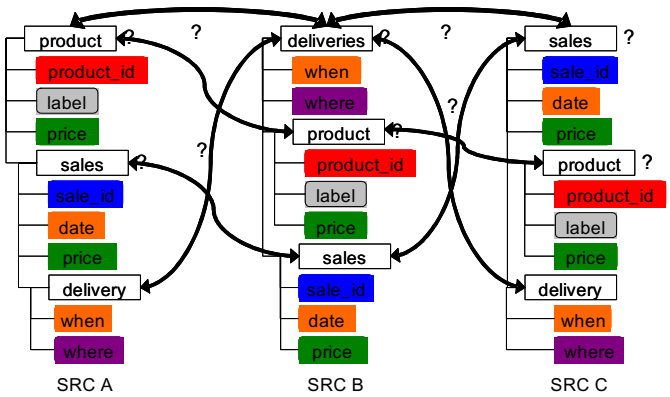

Fig. 7 Example of Schema Matching

product and sales information. SRC C is a sales department and hence sales data includes products and deliveries.

The shading of each leaf node indicates its domain. That is, when two leaf nodes share a shading it means that the system has determined that the two nodes derive their values from the same domain. The problem of matching is ambiguous. For example, considered as a whole, we might say that product in SRC A matches deliveries in SRC B and sales in SRC C, because all three structures contain the same information. On the other hand, it might be equally useful to record that product in SRC A matches deliveries/product in SRC B and sales/product in SRC C, because the information at each of these nodes is the same.

We start by constructing a schema tree for each data source using the parent-child relationship information that is recorded in the DataSet repository. We then construct a *flow network* for each pair of trees. One of the trees is labeled "source tree" and links inside it are oriented from parent to child. The other tree is labeled "sink tree" and its links are oriented from child to parent. These links are assigned an infinite flow capacity. A flow capacity of 1.0 is assigned to all those leaf-node to leaf-node pairs whose similarity exceeds a certain threshold. The Matcher computes the *Max-Flow* of the flow network, which indicates, for each node in the sink tree, the cumulative amount of flow from the source available at that node. Note that the flow at the root of the sink tree always matches the flow at the root of the source (flow preservation principle).

Similarity of two internal nodes n_A and n_B is calculated by the following formula:

$$\frac{MinCut(n_A, n_B)^2}{|Leaves(n_A)| * |Leaves(n_B)|}$$

where $MinCut(n_A, n_B)$ returns the flow from n_A to n_B in the flow network, $Leaves(n_A)$ returns number of leaves under n_A and $Leaves(n_B)$ returns the number of leaves under n_B. The number of leaves under an internal node is calculated in one of two ways, leading to two similarity metrics. In metric M1, all descendants of an internal node are included, while metric M2 includes only descendants that are not repeated. In Fig. 8a, the

member node in the sink tree has five leaves according to M1, but only four according to M2 (Fig. 8b), because the award node repeats (i.e., a player may have earned several awards).

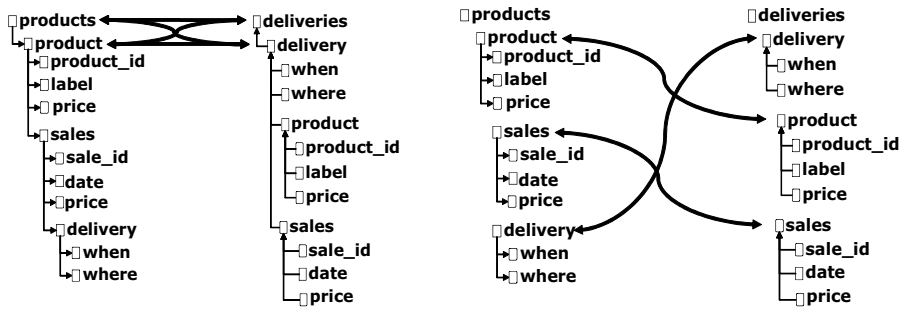

Fig. 8 Similarity of Internal Nodes according to Metric M1 (a) & M2 (b)

3.5 Discovering Constraints

In addition to computing similarity measures between data sets and more complex structures, the DBMS also discovers constraints within and between data sets such as *keys, relative keys, correlations, algebraic constraints*, and *functional dependencies*. As mentioned earlier, the data sets under consideration may come from the same data source or different data sources. The techniques for finding keys, key-to-foreign-key constraints, algebraic constraints, and correlations can be adapted from those in Brown and Haas [3] and Ilyas et al. [6]. In the current setting, the accuracy of some of these methods, which utilize distinct-value counts based on samples, can be improved: estimated counts based on a complete scan of the data can be derived from the hash-count data structure. For example, if the size of the domain of a data set (as estimated from the hash-count data structure) is close to the size of the data set itself (which is computed exactly), then the data set clearly satisfies a key constraint.

4 Identifying UBOs

Business objects are high-level data structures that reflect the business activity of an enterprise. For instance, a company engaged in selling of manufactured goods might care about business objects such as Customer, Order, Product, and Invoice. Inside each business object one may find other objects. For instance, an Order object may contain Product objects—products that a Customer ordered in that Order. We call these "nested objects". Alternatively, an object may contain a foreign key to another object. An Invoice object, for instance, could contain the ID number of the Order object it pertains to.

Business objects are important because they permit analysis and information integration at a higher level of abstraction. As we shall see in the next section, higher-level

business objects are also useful when interfacing with the user. The schema (definition) of business objects may be discovered automatically or it may be derived from an enterprise data dictionary. In large corporations it is not uncommon to find efforts to standardize various activities such as purchasing or sales across geographical and other divisions. An enterprise data dictionary is one of the objectives of such standardization efforts.

Universal Business Objects (UBOs) are those objects that are common to many data sources. More precisely, the *structure* of a UBO can be found in multiple sources. Consider the Customer UBO. Many sources contain customer information—order management systems, customer relationship management systems, marketing management systems, and so on. Customer information is used for different purposes but the basic structure is the same. Customer has attributes such as name, address, telephone numbers, and email address. Each source typically has additional attributes for the Customer. The order management system would probably contain the credit card number of the customer, while the CRM system would contain flags indicating the preferred contact method (phone, email, mail) for the customer.

UBOs are identified by computing a *degree of sharing* (DoS) measure for each schema structure and selecting the highest-scoring structures as candidate UBOs. DoS is computed from three inputs: source schemas expressed as leaf-level data elements and tree-like composite structures (as described in the *Dataset* repository), similarity of elementary and composite data structures across and within data sources (as described in the *Similarity* repository), and foreign key relationships across and within data sources. The system uses a number of heuristics to eliminate candidate objects that are unlikely to be UBOs. For instance, a candidate structure is probably not a UBO if only a single instance exists or if the structure is too small (e.g., city name) or too large (entire product catalog).

The UBO discovery process starts by considering each data source to be a single UBO and iteratively breaking large composite objects into smaller ones. A DoS score is computed for each object O as the sum of three components: *structural sharing*, *value relationships*, and *foreign key relationships*.

To compute a *structural sharing* score, the system looks at the links from O to its parent and superclasses—the more parents or superclasses O has, the higher the score. The link from O to its immediate parent(s) has a value of 1.0. Links to the parent's parents have a value of 0.5. Each level of ancestry has a value that is one-half of what the immediately lower level has. For instance, if O is 3 levels down from the root in a tree structure, it has a structural sharing score of 1 + 0.5 + 0.25 = 1.75.

Value relationship considers the similarity of O to other objects and uses the structural sharing value of those other objects to increase the score of O. For instance, if O is similar to object X (similarity value 0.8) and X has a structural sharing value of 1.5, then the overall score of O is incremented by 0.8 * 1.5. This is repeated for every object that O is similar to.

A *foreign key relationship* means that a specific instance of O (its key field) is referenced by another object X (its foreign key field). If the foreign key relationship has strength 0.9 and X has a structural sharing value of 1.75, the system increments the

overall score of O by 0.9 * 1.75. This is repeated for every foreign key relationship that points to O (its key).

Objects with a sufficiently high DoS score are marked candidate UBOs. Candidate UBOs are split off from their parent object and a foreign key is inserted into the parent (1:1 relationship) or into the UBO (1:N relationship). If the relationship between the parent and UBO is N:M, a separate relationship object is created. Similar UBOs are merged together, applying the intersection or union semantic.

To illustrate how UBO discovery works, consider the scenario depicted in Fig. 3. The system analyzes data from four sources (W, X, Y, and Z) and produces a synopsis for each of the data sets that they contain. The system also produces a synopsis for the reference data sets (ProdCategory, TIME, LOCATION). Similarities between the data sets in W, X, Y, and Z and the reference sets are computed at leaf node and internal node levels. A similarity is discovered between some pairs of data sources: W and LOCATION, X and ProdCategory, Y and TIME, and Z and LOCATION. A similarity is then calculated between W and Z, because both have some similarity to LOCATION, and it is determined that W is very similar to Z. Further analysis of the data sets reveals that both W and X are referenced from Y through a foreign key relationship. Object W has a high DoS score, because a very similar structure exists in another source (Z), it has some similarity to a reference data set (LOCATION), and it is referenced by another source (Y). Object W is merged with Z because the two have very similar structures, resulting in UBO W' (Customer) which is the intersection of the two objects. Object X is defined as UBO X' (Product) because it is referenced by Y and has similarity to the ProdCategory reference data set. UBO Y' (Order) is produced from Y and has a lower score than the other UBOs, because it only has similarity to one other data set (TIME).

5 Querying

In this section, we describe how a user can perform rich browse and search on our integrated schema and content repository. We first define the search space provided by our system. Then, we describe in detail how basic keyword search works and the nature of the returned results. Finally, we show how sophisticated search and analysis (BI) can be performed on the integrated data in terms of OLAP cubes and real-time dashboards.

5.1 Search Space

Searches in our integrated DBMS are specified in terms of UBOs (and not the underlying base data sets). In general, each UBO encapsulates a set of low-level data from various data sources. Conceptually, the search space is a graph of UBOs, where each node corresponds to either an *entity UBO* or a *relationship UBO*. Relationships represent "facts" in BI terminology. Fig. 9 shows an example of a search space. The example shows a graph with entity UBOs labeled "Product" and "Customer" that are connected through relationship UBOs labeled "Orders" or "Complaints". For illustrative purposes,

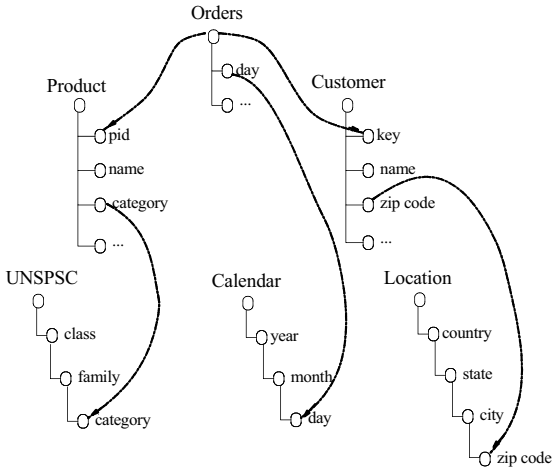

Fig. 9 Search Space Example

we have annotated the graph with UBO fields such as product name, customer key and order date. The percentages on the relationship links correspond to the "strength" of the relationship. In our example, 90% of Products relate to Orders and 40% of Products relate to Complaints. Similarly, 80% of Customers related to Order and 60% to Complaints. Additionally, the search space contains UBOs (*reference UBOs*) that correspond to reference data (like UNSPSC for products, Calendar for time, and Location for addresses).

5.2 Conceptual Keywords: Rich Browse, Search, and Ranking

The most basic form of searching that the system provides is the "conceptual" keyword based search, where keywords do not correspond to actual words in the underlying base data sets (as is the situation in typical text search engines) but to labels of entities or relationships between UBOs in the search space. We also assume that we have a synonyms catalog that contains synonyms for various keywords. For example, a synonym for the keyword "where" is "address", for "time" is "calendar", etc.

In order to explain how the conceptual keyword search works, let's assume that the user specifies the following query:

"product where when"

Using a synonyms catalog, the keywords are mapped to UBOs, as shown in the following table:

Keyword	UBO
Product	Product
Where	Location → Customer
When	Calendar → Order

In our example, keyword “Product” matches directly to Product, keyword “where” has a synonym “street” that matches through Location to Customer, and keyword “when” has a synonym for “day” that matches through Calendar to Order.

The system returns a query result graph similar to the one depicted in Fig. 3, along with some samples that are cached internally by our system. The user has the choice of seeing exactly how entities are related with each other and actual examples of such entities in the underlying data. The search can be refined, if necessary, by further constraining the result graph. For example, the user might not be interested in the UNSPSC categorization and can discard the corresponding part of the graph.

To handle the potentially huge number of subgraphs that match the query, the system can rank the matching subgraphs and display only the highest-ranked results. The ranking uses the similarity metrics that are computed during UBO discovery.

5.3 BI Support and Dashboarding

The metadata returned by the conceptual keyword search can be enhanced and exploited by the user to perform exploratory cube queries in an easy, intuitive way. Typically, the reference data sets contain (or can be easily enhanced with) well-known hierarchies such as time (day, week, month, year) or a standard hierarchical product categorization such as UNSPSC. The results of the basic keyword search are annotated by the system with such hierarchies. The user can designate certain attributes as measures (like the price of product) and certain entities as dimensions (like customer or product) just by clicking on the result graph. Operations like rollup or drilldown can then be easily performed by simply choosing the appropriate aggregation level in case of the hierarchies that are presented in the result graph. Because response time is very important for intensive BI applications, the performance for executing such queries must be optimized using techniques like pre-materializing certain parts of the cube using, for example, the techniques in [9].

Our system can also be used to support real-time dashboarding. Specifically, the user (via the intuitive data cube interface) can specify certain aggregations as “real-time”. The system then monitors the specified aggregated values and triggers appropriate actions when user-defined thresholds are crossed. In many cases, maintaining many such complex aggregates in real time requires approximation, and the techniques described in [10] can be applied. The results of BI reporting, data analysis, and real-time dashboards are stored using XML and interfaces to office productivity tools such as spreadsheets.

6 Conclusion

Modern businesses need to integrate thousands of highly heterogeneous information sources. These information sources are typically not tables of records, but rather collections of complex business objects such as purchase orders, invoices, and customer complaints. The traditional data warehouse approach of "manual" schema design and ETL is not feasible anymore, because this methodology does not scale and relies on expensive human resources.

We have outlined an approach to massive automated information integration that addresses these problems. Our framework exploits the enabling technologies of XML, web services, caching, and portals. Our approach also incorporates novel methods for automated similarity analysis that rest on synopsis-construction techniques such as sampling and hashing, pattern matching via automata, graph-analytic techniques for object identification, and automatic discovery of metadata for business objects. Our system provides a powerful querying interface that supports a wide variety of information-retrieval methods, including ad-hoc querying via intuitive keyword-based search, graph querying based on similarities, automatic report generation, graphical and traditional OLAP, spreadsheets and other office productivity tools, data mining algorithms, and dashboards.

Our system is far from complete. Important ongoing research issues include further improvement and empirical evaluation of data set signatures and similarity measures, scalability and deployment studies, more sophisticated methods for business object discovery, exploitation of ontologies, and incorporation of new query processing paradigms. The system must be able to exploit not only open standards such as XML, but also deal with domain-specific standards such as XBRL for financial solutions. In the long run, our system can potentially lead to an enormous enhancement of business productivity by providing users with a rich querying environment over information not only within the enterprise, but in the entire supply chain and beyond.

Acknowledgements

We thank Michalis Petropoulos for implementing the "Matcher" component in our prototype system, and Kai-Oliver Ketzer for implementing the incremental synopsis management.

References

[1] Angluin, D.: On the Complexity of Minimum Inference of Regular Sets, Information and Control, December 1978, 39(3), pp. 337-350

[2] Astrahan, M., Schkolnick, M., Whang, K-Y.: Approximating the Number of Unique Values of an Attribute without Sorting, Information Systems, 12(1), 11-15, 1987

[3] Brown, P. G., Haas, P. J.: BHUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data. Proc. 29th VLDB, pp. 668-679, 2003

[4] Carrasco, R. C., Oncina, J.: Learning Stochastic Regular Grammars by Means of a State Merging Method, Grammatical Inference and Applications, Second International Colloquium (ICGI), 1994, pp. 139-152

[5] Gibbons, P. B., Matias, Y.: New Sampling-Based Summary Statistics for Improving Approximate Query Answers. Proc. SIGMOD'98, pp. 331-342

[6] Ilyas, I., Markl, V., Haas, P. J., Brown, P. G., Aboulnaga, A.: CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. Proc. SIGMOD'04, pp. 647-658

[7] Poosala, V., Ioannidis, Y. E., Haas, P. J., Shekita, E. J.: Improved Histograms for Selectivity Estimation of Range Predicates. Proc. SIGMOD'96, pp. 294-305

[8] Pitt, L.: Inductive Inference, DFAs and Computational Complexity. 2nd Int. Workshop on Analogical and Inductive Inference (AII), 1989, pp. 18-44

[9] Sismanis, Y., Roussopoulos, N.: The Polynomial Complexity of Fully Materialized Coalesced Cubes. Proc. VLDB'04, pp. 540-551

[10] Sismanis, Y., Roussoupoulos, N.: Maintaining Implicated Statistics in Constrained Environments. Proc. ICDE'05

[11] Teradata Corporation: Getting it Together. Data Warehousing Report, 5(4), August, 2003. Online at: http://www.teradata.com