Integrating Relational Database Schemas using a Standardized Dictionary*

Ramon Lawrence
Advanced Database Systems Laboratory
University of Manitoba
Winnipeg, Manitoba, Canada
umlawren@cs.umanitoba.ca

Ken Barker
Advanced Database Systems Laboratory
University of Calgary
Calgary, Alberta, Canada
barker@cpsc.ucalgary.ca

ABSTRACT

Schema integration requires the resolution of naming, structural, and semantic conflicts. Currently, automatic schema integration is not possible. We propose that integration can be increasingly automated by capturing data semantics using a standardized dictionary.

Our integration architecture constructs an integrated view by automatically combining local views defined by independently expressing database semantics in XML documents using only a pre-defined dictionary as a binding between integration sites. The dictionary eliminates naming conflicts and reduces semantic conflicts. Structural conflicts are resolved at query-time by a query processor which translates from the semantic integrated view to structural queries. Thus, the system provides both logical and physical access transparency by mapping user queries on high-level concepts to schema elements in the underlying data sources. The architecture automatically integrates and transparently queries relational data sources, and its application of standardization to the integration problem is unique.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability; H.2.3 [**Database Management**]: Languages—*Data description languages, Query languages, Integration languages*; H.2.4 [**Database Management**]: Systems—*Relational databases, Multidatabases*

Keywords

Multidatabase, Relational schema integration, Automatic conflict resolution, XML, Standardization

*This research is partially sponsored by a NSERC Research Grant (RGP-0105566) and TRLabs

1. INTRODUCTION

Although numerous architectures provide database interoperability, automatic schema integration has not been previously possible. We propose that automatic schema integration is feasible by using a standard term dictionary to describe schema element semantics. Using the dictionary resolves naming problems and allows algorithms to automatically resolve structural conflicts in data representation. The major contribution of the work is a systemized method for capturing data semantics using a standardized dictionary and a model which uses this information to perform schema integration in relational databases.

This paper describes the integration architecture and compares it with existing work starting in Section 2. Section 3 overviews the components of our architecture including the standard dictionary, a metadata specification language, an integration algorithm, and a query processor. Although the architecture is in its infancy compared to more advanced mediator systems, it approaches the integration problem from a different perspective which may be more easily automated. A discussion of the architecture including its benefits and shortcomings is in Section 4. The paper closes with future work and conclusions.

2. DATA SEMANTICS AND THE INTEGRATION PROBLEM

Integrating data sources involves combining their concepts and knowledge into an integrated view that isolates users from the system organization. Constructing an integrated view of data sources is difficult because they will store different types of data, in varying formats, with different meanings, and will reference it using different names. Subsequently, constructing an integrated view requires resolving the different mechanisms for storing data (structural conflicts), for referencing data (naming conflicts), and for attributing meaning to the data (semantic conflicts).

Mediator and wrapper systems such as Information Manifold [4], Infomaster [2], and TSIMMIS [7] answer queries across a wide-range of data sources. TSIMMIS [7] and Infomaster [2] construct integrated views using designer-based approaches which are mapped using a query language or logical rules into views or queries on the individual data sources. Once an integrated global view and corresponding mappings to source views are logically encoded, wrapper systems are systematically able to query and provide interoperability between diverse data sources.

Internet and industrial standards organizations have taken a more pragmatic approach to integration by standardizing the definition, organization, and exchange mechanisms for data communications. Work on capturing metadata in industry has resulted in the formation of standardization bodies for exchanging data such as Electronic Data Interchange (EDI), Extensible Markup Language (XML [12]), and BizTalk [9] which allows data exchange using standardized XML schemas.

There is a fundamental difference between the database research approach and the industrial approach. Research algorithms analyze structural and semantic information to resolve schema conflicts. However, mostly manual algorithms have been developed because resolving all conflicts is extremely difficult. Thus, it has been proposed [1] that automatic schema integration is not possible. Industrial systems accept standardization to resolve conflicts and increase automation.

Our approach combines the two techniques. We believe that some level of standardization is required to achieve more automatic integration. Specifically, by accepting a standard term dictionary for describing schema element semantics and thus avoiding naming conflicts, structural conflicts may be automatically resolved. We approach the integration problem from a different perspective than mediator systems. Our goal is to separate the specification of database semantics from the integration procedure, and then apply automatic integration procedures to combine semantic specifications and resolve conflicts.

3. ARCHITECTURE COMPONENTS

There are four components of the architecture: a standard term dictionary, a metadata specification for capturing data semantics, an integration algorithm for combining metadata specifications into an integrated view, and a query processor for resolving structural conflicts at query-time. The dictionary provides a set of terms for describing schema elements and avoiding naming conflicts. The integration algorithm matches concepts to produce an integrated view, and the query processor translates a semantic query on the integrated view to structural query expressions.

To illustrate the architecture, we will use the following example involving two books databases. The first company, called **Books-for-Less**, has a database as given in Figure 1. The second company, called **Cheap Books**, stores its book database as given in Figure 2. Throughout the text, database field and table names appear in *italics* and their associated semantic names are in `true-type`.

Tables	Fields
Book	ISBN, Title, Author, Publisher, Price

Figure 1: Books-for-Less Database Schema

Tables	Fields
Book	ISBN, Author_id, Publisher_id, Title, Price, Description
Author	Id, Name
Publisher	Id, Name

Figure 2: Cheap Books Database Schema

3.1 A Standardized Global Dictionary

To provide a framework for exchanging knowledge, there must be a common language in which to describe the knowledge. Since a computer has no built-in mechanism for associating semantics to words and symbols, an on-line dictionary is required to allow the computer to determine semantically equivalent expressions.

The standard dictionary is organized as a hierarchy of concept terms. Concept terms are related using 'IS-A' relationships for modeling generalization and specialization and 'HAS-A' relationships to construct component relationships.

We have built a dictionary of terms starting from the top-level ontological categories proposed by Sowa [11]. For this paper, the simplified dictionary used is in Figure 3. Note that the exact terms and their placement is irrelevant. The dictionary is treated as a standard whether within an organization or for the whole Internet community. Individual organizations may modify the dictionary, but successful integration within a domain is only guaranteed with total standard acceptance. Thus, we assume that a designer correctly associates proper dictionary terms to represent schema element semantics, and mis-naming problems are handled using an external error-checking mechanism.

By analogy, the dictionary is like an English dictionary, as it defines the semantics of accepted words used to convey knowledge. However, overall semantics are communicated by organizing words into a structure such as sentences. Our structure for semantic communication is a *semantic name* whose simplified structure is easily parsed.

3.1.1 Constructing Semantic Names

A *semantic name* (*sname*) captures system-independent semantics of a schema element including contextual information by combining one or more dictionary terms as given in Definition 1.

Definition 1. $sname =$ "[" CT [[;CT] | [,CT]] "]" [CN]
where CT, CN are dictionary terms

That is, a semantic name consists of an ordered set of context terms (CT) separated by either a comma or a semicolon, and an optional concept name term (CN). Each context and concept term is a single term from the standardized dictionary. The comma between terms A and B (A,B) represents that term B is a subtype of term A. A semi-colon between terms A and B (A;B) means that term A HAS-A term B, or term B represents a concept that is part of term A. The context terms provide a context framework for the concept that describes them. Every semantic name has at least one context term. The concept name is a single, atomic term describing the lowest level semantics. Fields have concept names to represent their base meaning void of any context information. The semantic names for schema elements for Cheap Books and Books-for-Less are given in Figures 4 and 5 respectively.

Type	Semantic Name	System Name
Table	[Book]	Book
Field	[Book] ISBN	ISBN
Field	[Book] Title	Title
Field	[Book] Price	Price
Field	[Book;Author] Name	Author
Field	[Book;Publisher] Name	Publisher

Figure 4: Cheap Books Semantic Names

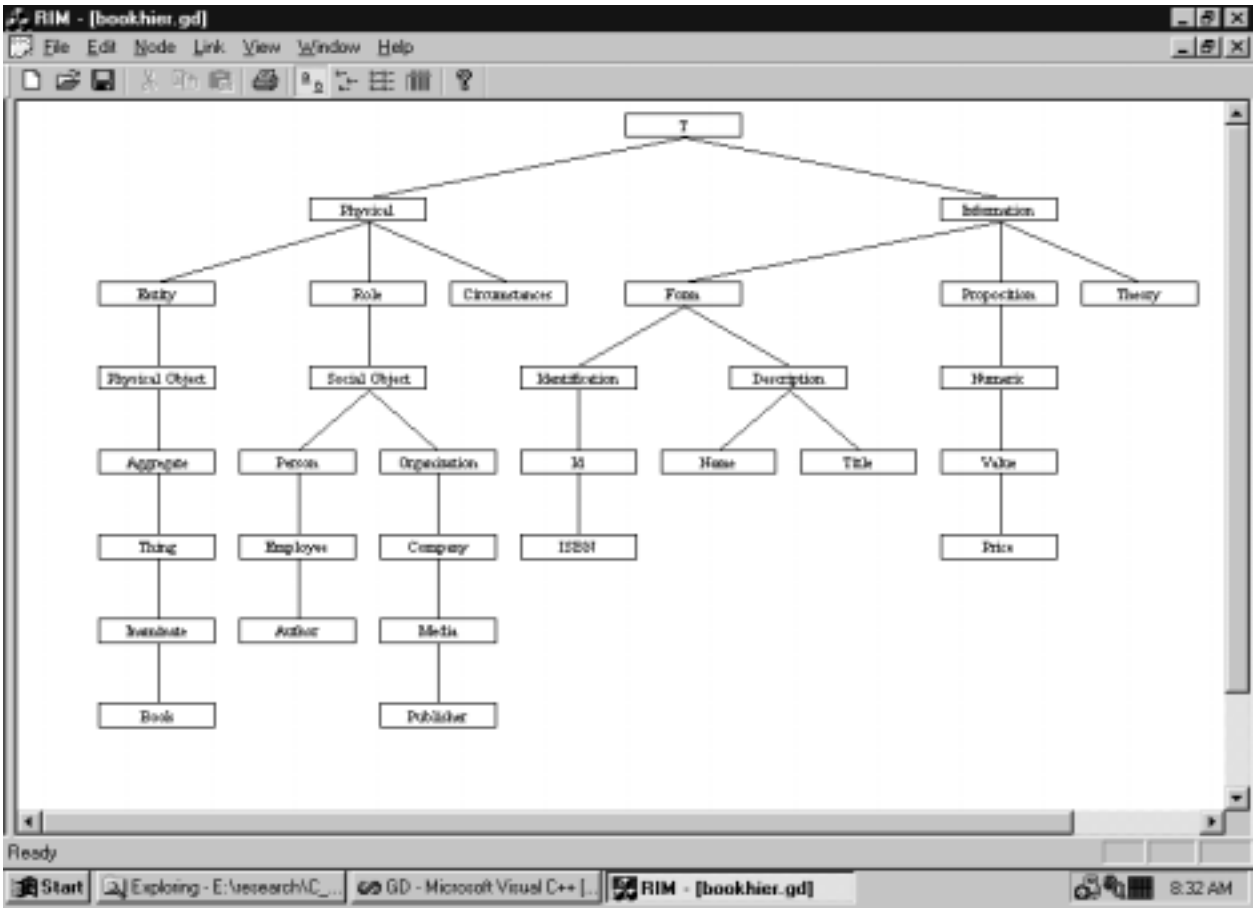

Figure 3: Reduced Standard Dictionary (contains only required terms)

Type	Semantic Name	System Name
Table	[Book]	Book
Field	[Book] ISBN	ISBN
Field	[Book] Title	Title
Field	[Book] Price	Price
Field	[Book] Description	Description
Field	[Book;Author] Id	Author_id
Field	[Book;Publisher] Id	Publisher_id
Table	[Book;Author]	Author
Field	[Book;Author] Id	Id
Field	[Book;Author] Name	Name
Table	[Book;Publisher]	Publisher
Field	[Book;Publisher] Id	Id
Field	[Book;Publisher] Name	Name

Figure 5: Books-for-Less Semantic Names

3.2 X-Specs for Metadata Specification

A standard dictionary is not a standard schema as concepts may be represented in different ways in various data sources, and we are not assuming a standard representation for a given concept. Thus, a XML-based specification document called a X-Spec encodes database schema using dictionary terms and additional metadata.

A X-Spec stores a relational database schema including keys, relationships, joins, and field semantics. Further, each table and field in the X-Spec has an associated semantic name as previously discussed. Information on joins including their cardinality, fields, and connecting tables is stored to allow the query processor to identify which joins to apply during query formulation. Similarly, field relational dependencies are stored for use in query-time normalization.

Describing a data source using a X-Spec is very similar to a standardized schema developed in BizTalk. We have made an attempt to follow emerging industry standards in the description of schemas using XML and model a X-Spec schema description after BizTalk schemas. The important distinction between a X-Spec schema and a BizTalk schema is that an entire BizTalk schema is standardized. A X-Spec describes a database dependent schema rather than conform to one. Simplified X-Specs for the two example databases are in Figures 6 and 7 which list the fields and tables and their semantic name mappings but omit the specification of keys and joins. In a X-Spec, the attribute *stype* stores "F" or "T" to indicate if the schema element is a field or table, and the attribute *sname* stores its system name.

A X-Spec is constructed using a specification editor during a capture process, where the semantics of schema elements are mapped to semantic names. This capture process is performed independently of capture processes at other data sources because the only "binding" between individual capture processes is the use of the dictionary to provide standardized terms for referencing data. We have built a specification editor that parses relational schema, formats the information into a X-Spec, and allows the user to include additional information that may not be electronically

```
<?xml version="1.0" ?>
<Schema
   name = "Books-for-Less.xml"
   xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">

<ElementType name="[Book]" sname="Book" stype="T">
   <element type="[Book] ISBN" sname="ISBN" stype="F"/>
   <element type="[Book] Title" sname="Title" stype="F"/>
   <element type="[Book] Price" sname="Price" stype="F"/>
   <element type="[Book;Author] Name" sname="Author" stype="F"/>
   <element type="[Book;Publisher] Name" sname="Publisher"
      stype="F"/>
</ElementType>
</Schema>
```

Figure 6: Books-for-Less X-Spec

```
<?xml version="1.0" ?>
<Schema
   name = "Cheap_Books.xml"
   xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">

<ElementType name="[Book]" sname="Book" stype="T">
   <element type="[Book] ISBN" sname="ISBN" stype="F"/>
   <element type="[Book] Title" sname="Title" stype="F"/>
   <element type="[Book] Price" sname="Price" stype="F"/>
   <element type="[Book] Description" sname="Description"
      stype="F"/>
   <element type="[Book;Author] Id" sname="Author_id" stype="F"/>
   <element type="[Book;Publisher] Id" sname="Publisher_id"
      stype="F"/>
</ElementType>

<ElementType name="[Book;Author]" sname="Author" stype="T">
   <element type="[Book;Author] Id  sname="Id" stype="F"/>
   <element type="[Book;Author] Name" sname="Name" stype="F"/>
</ElementType>

<ElementType name="[Book;Publisher]" sname="Publisher" stype="T">
   <element type="[Book;Publisher] Id" sname="Id" stype="F"/>
   <element type="[Book;Publisher] Name" sname="Name" stype="F"/>
</ElementType>
</Schema>
```

Figure 7: Cheap Books X-Spec

stored such as relationships and constraints.

The use of XML for describing an X-Spec is not required, but it is used because XML is an emerging standard to exchange semantics between systems. Information stored in a X-Spec may be transmitted as formatted text files or structured binary files. XML is used for convenience and interoperability with emerging standards. In summary, a X-Spec is a database schema and metadata encoded in XML that is exchanged between systems and stores semantic names to describe schema elements.

3.3 Integration Algorithm

The integration algorithm is a straightforward term matching algorithm. The same term in different X-Specs is assumed to represent the identical concept regardless of its representation. The algorithm receives as input one or more X-Specs and uses the semantic names present to match related concepts.

The integration process is automatic once the capture processes are completed. By their nature, capture processes are manual as they require designers to capture semantic information in X-Specs. However, once a capture process for a data source is completed, it never has to be re-performed

Global View Term	Data Source Mappings (not visible)
V (view root)	**N/A**
- [Book]	CB.Book, BfL.Book
- ISBN	CB.Book.ISBN, BfL.Book.ISBN
- Title	CB.Book.Title, BfL.Book.Title
- Price	CB.Book.Price, BfL.Book.Price
- Description	CB.Book.Description
- [Author]	CB.Author
- Id	CB.Book.Author_id, CB.Author.Id
- Name	CB.Author.Name, BfL.Book.Author
- [Publisher]	CB.Publisher
- Id	CB.Book.Publisher_id, CB.Publisher.Id
- Name	CB.Publisher.Name, BfL.Book.Publisher

Figure 8: Integrated View

regardless of the other data sources being integrated. This is a significant advantage as it allows database semantics to be captured at design-time. Thus, the advantage of the architecture is an integrated view is automatically created once designers independently define the local views of the individual data sources.

The integration algorithm identifies similar concepts by name regardless of their physical and logical representation and produces a hierarchy of contexts and concepts which implies no particular physical representation. The physical representation of concepts is irrelevant to the user. Users access data sources through semantic names which map to schema elements. Thus, by not imposing structural constraints on concept representation, knowledge is combined regardless of data representation characteristics, and the user is isolated from the complexities of data distribution, organization, structure, and local naming conventions.

The integration order is irrelevant, and the same X-Specs may be integrated several times with no change. As more X-Specs are integrated, the number of concepts grows, but assuming the semantic names are properly assigned, the effectiveness of the integration is unchanged. The view produced by integrating the Cheap Books (CB) and Books-for-Less (BfL) databases is given in Figure 8.

3.4 Query Processor

The integrated view of concepts is not a structural view consisting of relations and attributes. Rather, the *context view* is a hierarchy of concepts and contexts which map to physical tables and fields in the underlying data sources. Thus, querying the integrated view is different than existing systems, and implementing the query processor results in an entirely new set of challenges.

3.4.1 Query Formulation and Execution

Users generate queries by manipulating semantic names. The user is not responsible for determining schema element mappings, joins between tables in a given data source, or joins across data sources. The system handles the necessary joins based on the relationships between schema elements.

In many cases, there is a straightforward mapping from semantic names to physical fields. Typically, a semantic name will have only one mapping to a physical field in each data source. Given a list of semantic names in the query used either for projection or for selection criteria, the query processor maps the semantic names to system names using information stored in the X-Spec. To handle joins between tables, X-Specs store information on join conditions. Thus, all the required mapping information is present to construct a select-project-join query which is translated to SQL.

General joins across databases are not currently supported as query results from each data source are unioned together, unless the results can be combined using globally recognized keys such as a book ISBN or a product SKU number. If a given data source does not have all the fields required in the result, the field is left blank. Obviously, this method of query generation is simplistic. A more detailed treatment of query issues including join selection and optimization and SQL generation is available [5].

3.4.2 Query Examples

This section gives example queries on the integrated view and the SQL statements generated by the query processor.

Example 1. The user requires the ISBN (`[Book] ISBN`), title (`[Book] Title`), and description (`[Book] Description`) of all available books and selects the given semantic names from the context view.

Cheap Books	**Books-for-Less**
Select ISBN, Title	**Select** ISBN, Title, Description
From Book	**From** Book

Notice that no description field is available in the Cheap Books database. This field is left blank when a row result is displayed. The query system receives the output of both queries and displays them to the user. In this case, the query system would also attempt to match records based on the ISBN key because it is an internationally recognized key (not dependent on database context).

Example 2. The user requires all author names and so selects the semantic name `[Book;Author] Name`.

Cheap Books	**Books-for-Less**
Select Author	**Select** Name
From Book	**From** Author

In this example, a structural conflict is inherently resolved by mapping through the context view. The author name is retrieved from the *Author* table for Books-for-Less and from the *Book* table for Cheap Books.

Example 3. The user requires the title (`[Book] Title`) and author names (`[Book;Author] Name`) for all books.

Cheap Books	**Books-for-Less**
Select Title, Author	**Select** Title, Name
From Book	**From** Book, Author **Where**
	Author.id = Book.Author_id

In this case, the query processor must insert a join to process the query in the Cheap Books database. This join is automatically inserted because the system stores that *Author_id* is a foreign key to the *Author* table and a join can be applied to connect them.

As shown in these simple examples, physical and logical query transparency is provided to the user who queries the system by semantic name. The system handles the necessary mapping from semantics to a structural query and inserts join conditions as required. The challenge in this environment is discovering and constructing the query in an ad hoc basis based on the supplied mappings.

4. ARCHITECTURE DISCUSSION

The combination of standards such as XML and standard dictionaries with research algorithms in application to the schema integration problem is unique. Although schema integration and conflict resolution is well understood, there are no automatic algorithms for schema integration. The practical application of standardization to this problem enables architectures capable of more automated mediation.

The key benefit of the architecture is that the integration of data sources is automatic once the capture processes are completed. As discussed, a capture process is still a very manual process involving the designer thoroughly understand and record database semantics into a X-Spec. However, this process is performed only once and independently of all other capture processes. The only tie a capture process has to the global federation is the use of semantic names for identifying identical concepts across systems. This is similar to using a standardized set of XML tags except the dictionary terms may be used across integration domains. Then, the combination of X-Spec information is automatically performed and produces a integrated view of concepts which provides physical and logical transparency by allowing information access through a GUI based on semantics rather than structure. Structural conflicts are handled at query-time by the query processor without the user's involvement. This is a substantial improvement over systems [8] which require the user to query all databases by structure. Our work is unique because it automatically produces an integrated view from data source specifications developed independently of each other and the integrated view itself.

By accepting standardization, naming conflicts are eliminated and semantic conflicts are reduced by explicitly describing schema elements with standard terms. The architecture uses standardization to achieve automatic integration. Unlike Biztalk [9] or other E-commerce exchange initiatives, the architecture does not force a structural representation on the data which allows for greater flexibility and for existing systems to be unmodified during integration. Thus, the system preserves full autonomy of all data sources and no translational or wrapper software is required.

The major challenge inherent in the architecture is the definition of the standardized dictionary. Although defining terms to represent concepts is challenging, it is not without precedent. Industrial systems such as EDI, XML, and BizTalk all rely on the acceptance of standardized formats. Our architecture is even less restrictive as only names are standardized not structure and organization. Further, common ideas such as customers, orders, names, keys, identifiers, and addresses are well understood and easily mapped into a standard dictionary. In addition, even if a total standard is not achievable across the whole Internet, it is still possible to define localized standards. The architecture allows an organization to define its own dictionary. As long as the standard dictionary is conformed to within a domain, integration is possible within any organization. However, the ultimate goal is the definition of a standard dictionary applicable across all domains not just certain industries and environments as targeted by EDI, BizTalk, and E-commerce portals. Acceptance of standardization is a benefit, but it is not a common practice in the database community as it is difficult to acknowledge that some problems may require standardization to be solved.

Since the integrated view is constructed as needed with no designer input, challenges arise in insuring correct integrations. First, the architecture has no built-in mechanism for validating the assignment of semantic names. If a semantic name does not correctly capture the semantics of the schema element, it may be poorly integrated into the

integrated view. However, the concept is always present in the integrated view. Poor naming results from either poor conformance to the standard or inadequate construction of the X-Spec. Either problem can be resolved by re-examining and updating information in the X-Spec which will be automatically re-integrated into the integrated view.

Querying the context view produces an entirely new set of challenges. The system becomes responsible for mapping from semantic to structural expressions. Determination of the fields and joins is possible using the schema information in the X-Spec. However, complex integration challenges such as determining join conditions across databases and handling query-time normalization is an area of continuing work. Our ongoing focus is the definition of expanded query-time algorithms to resolve outstanding issues.

A related issue not covered by the architecture is data integration. Even though schema elements may be identical semantically, the actually physical representation in terms of types, sizes, currencies, and scaling factors may be different. There has been work performed on these data integration problems [3, 10]. Note that these problems are resolved by mapping functions which convert between contexts.

The integration architecture is implemented in a software package called Unity [6]. Unity allows for the construction and modification of the standard dictionary, automatic extraction of metadata into X-Specs, execution of the integration algorithm, and data source querying using the query processor and ODBC. Using Unity, integrating the Northwind database provided with Microsoft Access with another order-entry database was accomplished in less than a day. We continue to expand the functionality of Unity including refinement of the query processor.

5. FUTURE WORK AND CONCLUSIONS

In this paper, we have detailed a more pragmatic approach to schema integration. By accepting standardization in the form of a standard term dictionary, naming conflicts are eliminated and semantic conflicts are reduced. Database semantics are independently captured into XML documents called X-Specs which store semantic names for schema elements to identify identical concepts across systems. Then, an automatically constructed integrated view of concepts is transparently queried by the user. The query processor translates semantic queries to structural expressions and integrates results. Although the integration and query facilities are not as powerful as mediator architectures because they lack explicit designer control, by approaching the problem using standardization allows the integration to be performed more automatically.

6. REFERENCES

[1] B. Convent. Unsolvable problems related to the view integration approach. In *Proceedings of the International Conference on Database Theory*, pages 141–156, Sept. 1986.

[2] M. Genesereth, A. Keller, and O. Duschka. Infomaster: An information integration system. *SIGMOD Record*, 26(2):539–542, May 1997.

[3] C. Goh, S. Bresson, S. Madnich, and M. Siegel. Context Interchange: New features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–293, July 1999.

[4] T. Kirk, A. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *AAAI Spring Symposium on Information Gathering*, 1995.

[5] R. Lawrence and K. Barker. Multidatabase querying by context. In *DATASEM2000 - 20th annual conference on the Current Trends in Databases and Information Systems*, pages 127–136, Oct. 2000.

[6] R. Lawrence and K. Barker. Unity - A database integration tool. Technical Report TR-00-17, Department of Computer Science, University of Manitoba, July 2000.

[7] C. Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y. Papakonstantinou, J. Ullman, and M. Valiveti. Capability based mediation in TSIMMIS. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 564–566, June 1998.

[8] W. Litwin, L. Mark, and M. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22(3):267–293, Sept. 1990.

[9] Microsoft Corporation. BizTalk Framework 1.0 - Independent Document Specification. Technical report, Microsoft, Nov. 1999.

[10] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems*, 19(2):254–290, June 1994.

[11] J. F. Sowa. Top-level ontological categories. *International Journal of Human-Computer Studies*, 43:669–685, 1995.

[12] W3C. Extensible Markup Langauge (XML) 1.0. Technical report, Feb. 1998.