Flexible Semantic B2B Integration Using XML Specifications

Ken Barker
Computer Science, University of Calgary
Calgary, Alberta, Canada
barker@cpsc.ucalgary.ca

and

Ramon Lawrence
Computer Science, University of Iowa
Iowa City, Iowa, USA
ramon-lawrence@uiowa.edu

Abstract

XML has recently evolved as the key enabling technology to support B2B integration in the modern computing environment. Although much hype exists around the issue of data exchange between multiple businesses on the web, very few concrete proposals have evolved that are sufficiently flexible to exploit true data exchange. Two approaches have been proposed to facilitate data exchange from business to business. The first technique uses a standard exchange format to which all businesses wishing to participate in data exchange must conform. Alternatively, the businesses need to utilize a middleware solution that is sufficiently flexible to capture any type of data exchange but powerful enough to represent any data that needs to be exchanged. The system captures the key underlying semantics of the participating business data without forcing conformance to an inflexible standard. Further, since our system is based on XML, it exchanges data in a format that can be utilized by any application capable of reading XML data so both the schema and content are available for subsequent analysis.

Introduction

Much emphasis has recently been placed on developing suitable paradigms for conducting business-to-business (B2B) interactions. The needs of this application are inherently different than other integration models because they are formed on an as needed basis and often are *ad hoc*. Thus, traditional architectures such as multidatabase systems [1] are inappropriate because they introduce too much overhead since their primary goal is to provide integration across many database systems. Past work that is most similar to the B2B applications is the work undertaken on federated database systems [6] in that "one-off" integrations between two systems is defined. Each participant in a federated system must supply an *export schema*, which describes the data being made available, and an *import schema*, which describes the data being extracted from a remote system. Although the architecture is similar to that being developed for B2B applications, it suffers from the explosion of mappings that occurs each time a new system is incorporated into the federation.

Many industrial and research B2B applications propose the use of "standards" as the mechanism to facilitate communication. Several standards have been developed to support B2B data exchange including BizTalk [2], SIL [3], *etc*. Conformance to any standard is voluntary, but in many environments compliance is necessary to facilitate progress or even functionality. For example, network and inter-networking is only feasible if those participating are willing to follow the generally accepted standards. Support for network standards across organizations is also provided by common goals. Even if two organizations compete at all other levels it should be immediately evident that they can only both exist on the same "Internet" if they are willing to support a common network protocol.

Thus, the motivation for this work is based on the premise that although standards have a role to play in B2B interactions, they will never be a complete solution to the problem. This argument is based on two key facts. First, guaranteeing conformance to a standard will only work if one is defined for your particular application environment. Secondly, it would be impossible to anticipate all of the ways any two businesses might want to exchange data *a priori,* so a more flexible, possibly proprietary protocol, needs to be developed to enable data exchange.

Architectural Framework

Before discussing the details of our methodology, a B2B architecture is presented to frame the discussion. Figure 1 depicts two businesses that wish to exchange data across a network. A typical B2B scenario might be a supplier that provides some components required by a manufacturer to produce some product. The kinds of information that might need to be exchanged include order information (eg. POs, Sales Contacts, SKU Numbers, Shipping Dates, *etc*.), invoice information (eg. Invoice numbers, Contact information, Due Dates, *etc*.), and distribution channel information (eg. Shipper information (FedEx), delivery routing, tracking information *etc*.) The likelihood that both the shipper and receiver utilize homogeneous systems is extremely small so middleware is required to facilitate data exchange.

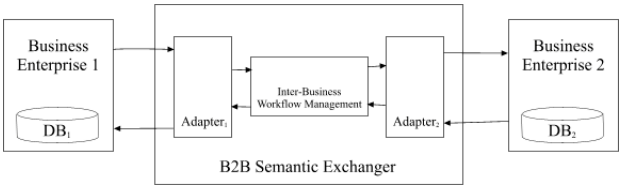

Figure 1: B2B System Architecture

To illustrate the B2B requirements, we will track an invoice sent from the supplier to the manufacturer depicted in Figure 1. We will make the unlikely assumption that the manufacturer wants to receive the invoice as quickly as possible and that it will be processed as soon as it arrives so the supplier can be paid immediately. To make the problem more realistic we will assume that the manufacturer's database is driven by Oracle while the supplier's is IBM's DB2. Both businesses will have their own schema and data formats and each will access their databases in quite different ways. It should be fairly obvious that each business will have an application that is capable of producing an invoice, processing it and ultimately ensuring that the transaction is completed successfully.

Thus, the supplier will generate an electronic invoice by extracting the required data from the data stored in its DB2

database. This will involve generating an internal invoice number, a description of the service supplied for the invoice, and a "e-packet" in the format used internally by the supplier's systems. This e-packet cannot be transmitted directly to the manufacturer because the format used by the manufacturer will likely be quite different. Thus, the manufacturer and supplier must agree on a common representation. If we assume that the two businesses agree on BizTalk as their exchange language, the supplier's e-packet must be converted to this format by the *adapter* before subsequent processing of the transaction can take place. In Figure 1 the processing of the invoice is accomplished using the module labelled *Inter-Business Workflow Management* where the "business logic" is performed. Executing this workflow will undoubtedly require data and/or permissions from the manufacturer, but this can only be achieved if the request is translated from the BizTalk format to the native format processable at the manufacturer's site. This requires another adaptor capable of translating the request into an e-packet capable of executing the necessary transaction on the manufacturer's machine. Once this has been processed, the entire process must now be reversed to move the answer, hopefully an acknowledgement of the receipt and payment of the invoice. This requires that another e-packet (native to Oracle) be produced, which is then translated to BizTalk for further workflow processing. Finally, the result of this intermediate processing is then translated back to a format understood by the supplier's DB2 database before a final acknowledgement of the transaction can be made.

Contributions

Based on this scenario we can now describe where this paper contributes. We are primarily interested in the middle component of Figure 1. Further, we are primarily interested in the technology necessary to facilitate data transmission through this middle component. The details of the workflow are clearly key to any B2B application, but our focus is not primarily on how to write this business logic. Rather, we are interested in the suitability of various techniques to exchange data from one business (the supplier) through the middleware component to another business (the manufacturer). We argue that standardization efforts aimed at developing a universally accepted *lingua franca* for such high level B2B applications will never be fully successful. These systems are too rigid to adapt to unanticipated application needs and will ultimately be ignored by businesses because of the costs of changing previously built applications that no longer conform to the new standard. As proof of this claim, consider the enormous time lag between the proposed change from IPv4 to IPv6 in the network community despite the nearly universal acknowledgement of the need for the update. Instead we argue that successful data exchange is only achievable if a system is provided that is readily extendable to new application needs while providing a framework that ensures participants easily conform to the *lingua franca*. Thus, our work focuses on developing an extensible exchange "language" that captures the semantics of the businesses that are willing to conform to the lingo.

To this end we propose a system capable of capturing the specific data needs for individual businesses. Although we do not believe it will be ultimately necessary, it is possible for our system to define specific exchange schemas between any two businesses in much the same way as was initially proposed for federated systems. Thus, individual businesses can use our tool to write specific "wrappers" for each of their systems. However, based on substantial experience, we know that there is an enormous amount of overlap between businesses. This is particularly true when you consider who is likely to participate in a B2B exchange. To illustrate this point consider the following simple example. Select any major book retailer (Barnes and Noble or Chapters) and consider for a moment with whom they are likely to undertake a B2B transaction. Clearly the answer is a related business such as a book supplier (Morgan Kaufmann or Wiley Press). It is extremely unlikely that a pharmaceutical business will undertake a B2B transaction with the book retailer to sell drugs. Thus, businesses in the book industry are likely to have a common linguo that is used by all participants and the slight differences, that will undoubtedly exist, can be readily addressed using the system described shortly.

We can now consider the key elements that must exist to facilitate this exchange. First, the core of any conversation is the need to have both participants speak a common language. Booksellers are able to undertake a dialog because they have common terms for common concepts. Thus, the first element is a "standard dictionary" that can be used to define what term is used to represent what concept. Unfortunately, data exchange between businesses is never based on using precisely the same terms to represent the same semantic in both organizations. Thus, we must be able to map from the syntax used to represent the concept at an organization to its representation in the standard dictionary. Fortunately, we only need to do this once for each business and it is interesting to note that the mapping from the standard dictionary term to the syntax used at the business is simply an inverse of the first mapping. Once the dictionary is defined and the mappings are in place, executing transactions between the two businesses requires that the business workflow for all such transactions be written. The architecture presented here is inherently different than past proposals because the business logic is not accomplished using the language of the legacy systems at each business but rather by using the standard dictionary's terms and concepts so applications developers can manipulate an integrated view of the data for the first time.

The balance of the paper describes the process for creating the standard dictionary by presenting an architecture that describes the capture process. Selection of the underlying representation language is always a critical decision when developing any system suitable for integrating legacy systems. The problem is further complicated because no matter what selection is made, it too, will ultimately be another legacy system. Thus, we want to select a environment that will likely have the longest possible life looking into the immediate future but, more importantly, is sufficiently extensible to allow it to adapt to changing needs into the more distant future. Thus, we have selected XML as the implementation language because of its inherently eXtensible nature. These issues in addition to our capture process to create the standard dictionary are detailed in the section describing Unity's architecture.. The integration of a business' data source into the system is accomplished by defining mappings using common terms that are used by the middleware, and is detailed afterward. The final technical

element described in this paper details how queries are accomplished using the middeware described. Clearly, the key element of the workflow management component depicted in Figure 1 is only feasible if queries can be posed by one system, translated to the common language, and posed at the other business. Once queries can be asked and answered it should be evident that the system is capable of providing the middleware necessary for B2B processes. Next a query process is described to detail the query processing features of our middleware and provides an example of its utility. The penultimate section provides a very brief review of other research activities leaving the final section to summarize our insights and provide pointers for subsequent research.

Unity Architecture

As mentioned above, the adapters depicted in Figure 1 are the focus of the work reported here. Although it would be tempting to consider these little more than wrappers for the participating business' database, the system is actually much more powerful. Unlike wrappers, these adapters must also provide facilities to define the underlying ontology often provided by standards conformance. This requires a representation of the ontology, the software to produce an arbitrary ontology, translation mechanisms to a flexible semantic notation, and a query processing capability so results can be exchanged from B2B. These adapters are the essence of our middleware solution, which is called Unity[1] to reflect a goal of providing a unified data exchange mechanism. This section explores some of the details associated with Unity.

The Unity architecture consists of five main components: a standard term dictionary, a dialect of XML used to specify metadata (X-Specs) that captures data semantics, an integration algorithm for combining X-Specs into an integrated view, a query processor for resolving conflicts at query-time, and "wrapper" software at each database site responsible for accessing participating databases available at businesses. The dictionary provides terms for describing schema elements and avoiding naming conflicts thereby forming an unambiguous *lingua franca*. The integration algorithm matches concepts from X-Specs to produce an integrated view, and the query processor translates a semantic query on the integrated view in the dialog expected by the business receiving the request. The wrapper software verifies user access to the system, processes SQL requests, and returns results.

The architecture utilizes three component processes:

- **Capture Process:** A capture process is independently performed at each data source to extract database metadata into a XML document called a X-Spec.
- **Integration Process:** The integration process retrieves X-Specs from each database and combines X-Specs into a structurally-neutral hierarchy of database concepts called an integrated context view (see Figure 2-a).
- **Query Process:** The user formulates queries on the integrated view that are mapped by the query processor to SQL. The SQL is transmitted to each database wrapper. The results returned are integrated and formatted (see Figure 2-b).

To illustrate the architecture, we use the following example involving two book databases. The first company, called **Books-for-Less**, has a database as given in Figure 3. The second company, called **Cheap Books**, stores its database as described in Figure 4. Note that database field and database names appear *italics* and semantic names in the integrated view are in Arial Narrow.

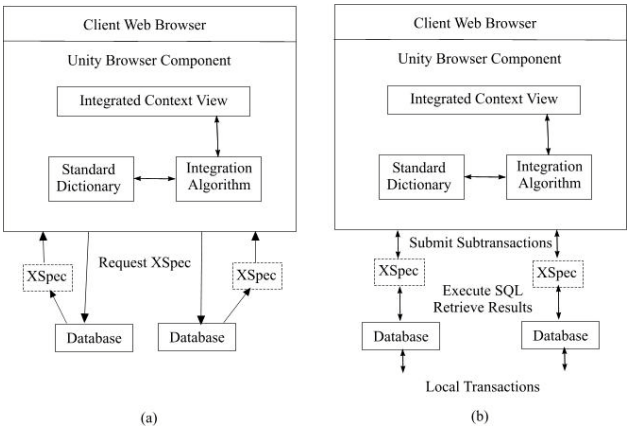

Figure 2: (a) The Integration Process of Unity; (b) The Query Process in Unity

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

Figure 3: Books-for-Less Database Schema

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

Figure 4: Cheap Books Database Schema

The Capture Process

The capture process is an off-line procedure where the semantics of a relational database schema are captured into a XML document called a X-Spec. The X-Spec is designed to store sufficient schema metadata such that it can be compared and integrated across systems. Using a standard term dictionary allows related concepts to be uniquely identified by name.

Standard Dictionary

The foundation of the architecture is the acceptance of a standard term dictionary which provides terms to represent concept semantics that are agreed upon across systems. Thus, the architecture operates under the assumption that naming

[1] Unity is a proprietary system developed at the University of Manitoba.

conflicts are prevented by utilizing standard terms to exchange semantics. Without a standard set of terms or names to communicate knowledge, knowledge cannot be integrated or exchanged because its semantics are not known. Thus, by accepting a standard dictionary, schema, or set of XML tags, a system assumes away the naming problem by accepting a lexical semantic framework for the expression of data semantics similar to our human acceptance of spoken languages to facilitate communication.

The standard dictionary is 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.

Constructing Semantic Names

A *semantic name* captures system-independent semantics of a relational schema element by combining dictionary terms. In the relational model, a semantic name is a **context** if it is associated with a table and a **concept** if it is associated with a field. A context contains no data itself and is described using one or more concepts. A semantic name, which is a concept, represents atomic or lowest-level semantics. In relation to the object-oriented model, a context is like an object, and a concept is an attribute of an object.

A semantic name consists of a context and concept portion. The context portion is one or more terms from the dictionary, which describe the context of the schema element. Adjacent context terms are related by either IS-A (represented using a ",") or HAS-A (represented using a ";") relationships. The concept portion is a single dictionary term called a concept name and is only present if the semantic name is a concept (maps to a field). The formal specification of a semantic name (*sname*) is:

sname ::= [CTerm] |
[CTerm] <CN>
CTerm ::= <CT> |
<CT> ; CTerm |
<CT> , CTerm

where CT and CT are dictionary terms.

The semantic names for **Books-for-Less** and **Cheap Books** are given in Figure 5 and Figure 6, respectively.

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

Figure 5: Books-for-Less Semantic Names

Type	Semantic Name	System Name
Table	[Book]	Book
Field	[Book] ISBN	ISBN
Field	[Book] Quantity	Quantity
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 6: Cheap Books Semantic Names

X-Spec - A Metadata Specification Language

A X-Spec is a XML-based specification document which encodes relational database schema information using dictionary terms and metadata including keys, relationships, joins, and field semantics. Further, each table and field has a semantic name as previously discussed. Metadata information on joins and dependencies are stored for query processing. A X-Spec is constructed using the specification editor component of Unity during the capture process.

The Integration Process – Forming the Ontology

The integration process combines the X-Specs retrieved from each data source into an integrated context view. The integration algorithm is a straightforward term matching algorithm. The same term in different X-Specs represents the identical concept regardless of its format. The algorithm receives as input one or more X-Specs and uses the semantic names present to match related concepts. 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 integrated view produced for the book databases is given in Figure 7.

Global View Trm	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
Quantity	CB.Book.Qty,BfL.Book.Quantity
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 7: Integrated View

The Query Process

The integrated view of concepts, called a *context view*, is a hierarchy of concepts and contexts, which map to physical tables and fields in the underlying databases. Businesses can query each others' repositories by generating queries by manipulating semantic names. The querying business is not responsible for determining schema element mappings, joins between tables in a data source, or joins across data sources. The system inserts joins based on the relationships between schema elements.

The query processor in Unity:

- Determines the semantic names of concepts requested by the query, and for each data source, determines the best field mapping(s) for each semantic name and their associated tables.
- Given a set of fields and tables to access in a data source, determines which joins to insert to connect database tables.
- Generates SQL queries created in the previous steps, and transmits SQL queries and authentication information to the wrapper systems for each data source.
- Retrieves row results from wrapper systems, applies reverse mappings back to semantic names, and displays formatted results to the query poser.
- Determines if row results should be unioned or joined together across databases based on the presence of common keys.

Query Example

Given the two bookstores described above, we now consider a typical e-business scenario. A third book retailer (**FindAll Books**) needs to locate as many copies of a book entitled "*How to Query Databases*" as possible. The first step is to integrate the **FindAll**'s business into the ontology described earlier.[2] This requires the creation of an X-Spec so the results, once located, can be returned. The integration algorithm must integrate **FindAll**'s X-Spec into the integrated dictionary to form the ontology. **FindAll** can then submit a query based on the integrated schema so it can retrieve the necessary information from both stores. Authentication and security access codes for each business' database are required, but this is the responsibility of the Workflow Management component of Figure 1. For each database, this information is stored in Unity (the adapter) so information such as its website address and authentication information can be retrieved and transmitted automatically.

[2] This is not strictly required if the only business requirement is to retrieve data from the two bookstores. However, if business workflow is required between the three businesses, the workflow manager (see Figure 1) must understand the ontological model of all participants

FindAll now selects the quantity available of the book entitled ``How to Query Databases'' using the integrated context view illustrated in Figure 7. Thus, two attributes are required from the integrated ontology:

[Book] Title = "How to Query Databases"
[Book] Quantity

The query processor now uses the mappings described in Figure 7 to produce the following SQL for each data source, which are sent to the adapters (recall Figure 1) for submission to each of the bookstore databases for processing:

Cheap Books	**Books-for-Less**
Select Qty **From** Book **Where** Title = "How to \ Query Databases";	**Select** Quantity **From** Book **Where** Title = "How to \ Query Databases";

This wrapper then returns results to Unity, which subsequently follows the directions of the workflow manager to return the integrated results to **FindAll**. Purchasing the book copies would require additional business logic that would need to be placed into the workflow manager, but **FindAll** does not need to do anything further for subsequent data exchange with **Cheap Books** or **Books-for-Less**. Clearly, this is an extremely powerful data interchange paradigm.

Related Work and Architecture Discussion

Mediator and wrapper systems such as Information Manifold [4] and TSIMMIS [5] answer queries across a wide-range of data sources. These systems construct integrated views using designer-based approaches, which are mapped using a query language or logical rules into views or queries on the data sources. Once an integrated view and corresponding mappings to source views are logically encoded, wrapper systems are systematically able to query and provide interoperability between data sources.

Internet and industrial standards organizations take 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 [7]), and BizTalk [2]. Industrial systems achieve increased automation by accepting standards to resolve conflicts.

Unity combines standardization with algorithms for conflict resolution. By separating the specification of database semantics from the integration procedure, Unity implements automatic procedures to combine specifications and resolve conflicts. The combination of standardization with research algorithms to address the schema integration problem is unique.

The key benefit of the architecture is that the integration of data sources is automatic once the capture processes are completed. By their nature, capture processes are partially manual, as they require designers to capture semantic information in X-Specs

using the X-Spec editor. Once a capture process for a data source is completed, it never has to be re-performed. Thus, the advantage of the architecture is a global view is automatically created once designers independently define the local views of the individual data sources. Further, Unity preserves full autonomy of all data sources.

The major challenge inherent in the architecture is creating the standard dictionary. Although defining terms to represent concepts is challenging, it is not without precedent. Industrial systems such as XML and BizTalk all rely on the acceptance of standard formats. Our architecture is even less restrictive as names are standardized but not structure.

Unity achieves automatic conflict resolution by using a standard dictionary to build semantic names, constructing a structurally-neutral integrated view from semantic names, and mapping semantic queries to SQL. The standard dictionary resolves the table naming conflict and the attribute naming conflict because contexts (tables) and concepts (attributes) will not be integrated unless they have the same semantics. Structural conflicts are resolved by mapping queries through the integrated view. Data level conflicts are resolvable by defining functions, which convert between contexts, and by formally expressing context semantics.

Future Work and Conclusions

This paper has described Unity's ability to support the important environment commonly referred to as "B2B". By utilizing Unity's philosophy of combining standardization and *ad hoc* schema integration, an extremely powerful paradigm is achieved. Thus, the user is able to create the key component to data exchange in the B2B environment, namely an ontology for data exchange. This ontology is based on the database semantics independently captured using the emerging XML language to exchange data between businesses. The paper has illustrated some of the power of X-Specs, which store semantic names for schema elements thereby identifying identical concepts across systems. We also illustrated how the integrated schema is mapped to queries at the participating databases and returned results used for subsequent business workflow.

Unity is not yet a complete work. The current implementation has shown utility in multidatabase and datawarehouse environments, but these are predominantly characterized by being "read-only". The B2B environment will require support for updates at multiple data sources. Although we believe support for updates in Unity should be a fairly easy extension for a single database, it is likely to prove quite challenging for an arbitrary business workflow that must atomically update multiple data sources. Thus, we are investigating transaction support for Unity in a B2B environment.

Bibliography

[1] M.W. Bright, A.R. Hurson, and S.H. Pakzad, "A Taxonomy and Current Issues in Multidatabase Systems", *IEEE Computer,* 25(3):50-60, March 1992.

[2] Microsoft Corporatation, "BizTalk Framework 1.0 – Independent Document Specification", Technical Report, Microsoft, November 1999.

[3] Uniform Code Council Inc., "SIL – Standard Interchange Language", Technical Report, January, 1999.

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

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

[6] A. Sheth and J. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases", *ACM Computing Surveys,* 22(3):183-236, September, 1990.

[7] W3C, "Extensible Markup Language (XML) 1.0", Technical Report, February 1998.