INFORMATICA, 2009, Vol. 20, No. 3, 417–438

XML in Enterprise Systems

Jaroslav POKORNÝ
Faculty of Mathematics and Physics, Charles University
Malostranske nam. 25, Praha, Czech Republic
e-mail: pokorny@ksi.mff.cuni.cz

Received: September 2008; accepted: March 2009

Abstract. The widespread use of the XML format for document representation and message exchange has influenced techniques for data integration in last years. A development of various XML languages, methods and tools gave rise to so called XML technology. Enterprise Information Integration (EII) requires an accurate, precise and complete understanding of the disparate data sources, the needs of the information consumers, and how these map to the business concepts of the enterprise. Any integration takes place in context of an Enterprise Information System (EIS). In the paper we explain various approaches to EII, its architectures as well as its association to Enterprise Application Integration (EAI). We introduce basic features and issues of EII and justify why XML technology contributes to finding sufficiently powerful support for tools for enabling EII. In particular, a database approach to XML provides a universal solution enabling to construct tools for achieving EII. In the paper we present some features of the XML technology, mainly its database part, and show how it is usable in EII.

Keywords: XML, enterprise information integration, XQuery, XSLT, Web services, XML databases.

1. Introduction

The language XML (W3C, 2004c), originally designed as a standard protocol for data exchange, serves today as a data model and background for databases of XML documents. Its main advantage is that it enables to create a background for applications beyond conventional data models, i.e., everywhere where we need, e.g., hierarchical data structures, recursive data structures, regular expressions, missing and/or duplicate data, and other non-traditional data requirements. At last but not least, XML creates a technological platform for Semantic Web (e.g., Antoniou and van Harmelen, 2004).

A collection of languages, techniques, and standards developed by the World Wide Web Consortium (W3C), called an *XML technology* today, contributes to many application areas, as, e.g., B2B interactions, Web services, as well as, in general, to improvement of inter- and intra-enterprise applications. In the paper, we focus just on use of XML technology in enterprises.

Often an *Enterprise Information System* (EIS) is characterized as an information and reporting tool for the preparation, visualization, and analysis of operational enterprise data. An associated collection of activities and software components supporting accessing data from any source systems is then called *Enterprise Information Integration* (EII).

In other words, EII provides programmers with a single-site image of disparate data that may be maintained in different formats, retrieved via different Application Programming Interfaces (APIs), and managed by different remote servers. The analyst community and other observers talk often about "virtual data federation". Data integration is crucial in large enterprises that own a multitude of data sources, like relational databases, Web services, files, and packaged applications. The same holds for offering good search capabilities across amounts of data sources on the Web.

EII is even considered as an industry today. By Halevy *et al.* (2006), factors positively contributing to this industry include occurrence of some matured integration technologies, change of the needs of data management (creating Web sites integrating data from multiple sources), and the emergence of XML, which subsequently causes an increased need of people to share data. Other significant motivation for EII is the growth of unstructured and semi-structured repositories and the convergence of the data and content worlds.

Integration-related area contains also *Enterprise Application Integration* (EAI). EAI integrates application systems by allowing them to communicate and exchange business transactions, messages, and data with each other using standard interfaces. It enables applications to access data transparently without knowing its location or format. EAI is usually employed for real-time operational business transaction processing. It supports a data propagation approach to data integration. A strong separation of EII and EAI can mean that enterprise data is accessed by an EII tool and updated by an EAI tool. EII solutions today should address both application and information integration.

In general, EII needs (Gilbane, 2004) to:

- support all information types, structured, unstructured and semi-structured,
- provide for context, i.e., where does the information fit in the schema or ontology of the receiving repository/application, and what are the relevant behavioural constraints.

Many enterprises today are moving towards the adoption of *Service-Oriented Architectures* (SOAs) based on XML and Web services (Fremantle *et al.*, 2002). Web services represent a less costly and loosely-coupled approach for EII. Often Web services are considered as part of the EII whole. Consequently, a *service composition* gains strength in EIS today. A relatively little work has been done to facilitate integration at the *presentation* level, i.e., the development of user interfaces. This part of application development in EIS belongs to the most time-consuming activities. Other direction of the EAI industry is toward the use of an *Enterprise Service Bus* (ESB) that supports the interconnection of legacy and packaged applications, and also Web services.

XML, enabling to declare and enforce structure of content, plays an important role both in EIS development and EII processes. The reason is simple. In the past, any exchanging information between content repositories and data-oriented applications within and across enterprise was very difficult due to incompatibility of supporting systems. Even data warehouse solutions were considered inappropriate for supporting such needs. EII vision is namely to provide tools for integrating data from multiple sources without first loading their data into a central warehouse. EII should perform the integration in real time on an as-needed basis.

Emergence of XML made it possible to build EII on an XML data model and query language XQuery (W3C, 2005), i.e., with XQuery interface to these multiple sources. The success of XML is directly related to the information integration problem. It was only a few years ago that the majority of IT industry was looking at XML primarily as a tool for sharing data and messages for EAI applications despite of the fact that XML technology has been less mature and required a shift rather toward XML-centric application development. XML was also recognized as being equally suitable for structured and unstructured data, and, as a way to connect legacy repositories with the future growth of more sophisticated repositories either based on XML or easily encoded in XML. XML offered a common syntactic format for sharing data among data sources.

The purpose of this work is to summarize some parts of the XML technology relevant for EII and show, how XML databases can help to create more responsive EII architectures. The remainder of the paper is organized as follows: Section 2 mentions some approaches to EII as well as commercial products based on these approaches. After summarizing some basics of XML technology in Section 3, in Section 4 we discuss a role of a database approach to XML, i.e., we present how the basic database notions, like schemas and query languages, are reflected in XML technology. We also introduce today's architectures of XML databases. Section 5 focuses on applications of XML databases, in particular of native ones, in EII. Particularly we mention the concept of content management systems. Section 6 offers some open questions for research and development. Finally, Section 7 concludes and lists future work.

2. EII: Motivations and Related Works

Data integration has been studied around for the longest period of time. There are two main approaches to data integration:

- local architecture (data is imported to one database, typically a warehouse),
- distributed (data remains in remote sources).

The former is the traditional approach to data integration, often called *Extraction, Transformation and Loading* (ETL). Data from disparate systems are accumulated into a single data store for the purposes of manipulation and evaluation (decision support). Data warehouses and data marts are the data stores, and ETL tools are the "data integration" components. Typically, data integration addresses only structured information. The most often examples of data integration include data processing in areas like health, flights, customer support, financial data, and products. Techniques of integration are based on:

- addressing differences in the query,
- conversion all data to the same schema,
- building common indexes over data.

2.1. *EII: Approaches to Integration*

Often spontaneous development of data stores and applications in enterprise results in a need of information integration. To do it in the traditional way, one has to rely on

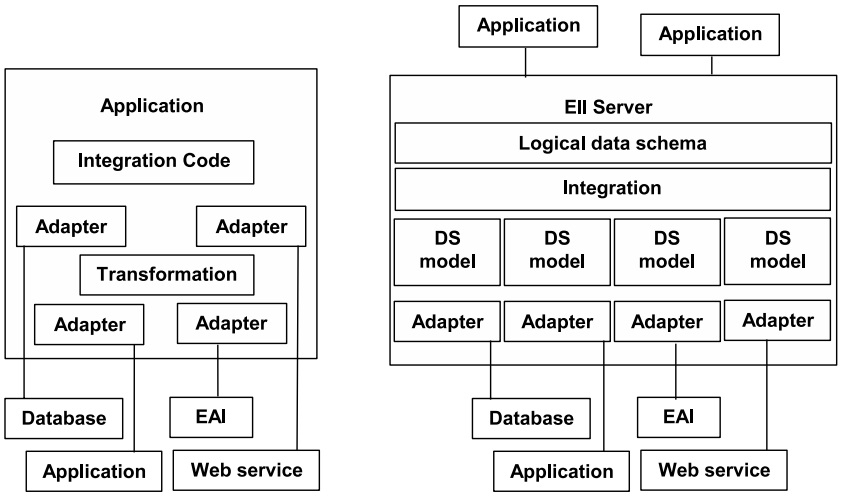

Fig. 1. Traditional vs. EII approach to data integration.

adapters for accessing data sources, transformation and integration software, and data warehouses (the left part of Fig. 1). The deficiencies of this approach include mainly low-level programming, detail knowledge of data sources, and very limited reusability of designed data schemas and integration rules.

By Tailor (2004), EII is the integration of data from multiple systems into a unified, consistent and accurate representation geared toward the viewing and manipulation of the data. Data is aggregated, restructured and relabelled (if necessary) and presented to the user. In viewing EII from a software engineering point of view, it is a type of middleware that allows companies to combine data from disparate sources into a single application.

EII is based on a more flexible form of integration than simple data integration. In EII data sources are viewed by applications as a single virtual database (the right part of Fig. 1b). EII is based on a framework that exposes rather declarative interface for specification of integration requirements. EII provides applications a single, *virtual view* across multiple data sources. Applications access data sources through these views and through only one API of the EII server. Queries are transformed into queries against data sources. In other terminology, this approach is based on mediation. There is also a variant called *federation*, which integrates data by defining mappings between all pairs of schemas of the member databases. This variant called also a *loosely coupled* federated system is not broadly applied in real enterprise environment because of the using of private protocol and data model, low performance, laborious process, critical implementation conditions, immature technology and the lack of reliable infrastructure (Zhou *et al.*, 2006). Although the federated systems are relatively easy to implement, they do not scale well. By Mattos (2003), an information integration infrastructure should support placing and managing data at multiple points in the data hierarchy to improve performance and availability. By the way, most of today's EII systems are really federated information systems.

To implement mediation, EII requires an accurate, precise and complete understanding of the disparate data sources, the needs of the information consumers, and how data model is mapped into a single, generic representation – a *logical data schema* that specifies the virtual view. A view represents a *business entity* – a customer, a sales pipeline, or the performance of a manufacturer's production floor – annotated with metadata. *Business metadata* includes business definitions, descriptions, and explanations relating to the view and its intended use. *IT metadata* describes how information gets from the various sources to its destination, including locations of data sources, data transformations, mappings, joins, and target schemas. Both the schema development and mappings of data sources use usually a GUI tool.

2.2. *EII: Architectures and Products*

Therefore, the available approaches to EII can be considered based on the underlying logical model, the data transformation framework, and the query interface.

Logical models include relational, object-oriented, and XML. The relational solution reminds distributed relational databases from 90ties. Due to the well-known restrictions of relational data model in context of enterprise variety of data, it is not too perspective solution now. For example, iWay Data Hub[1] enables to create reusable relational views. In the object-oriented approach all data sources are represented as objects, and automatically generated code is used to transform data into the logical model. Both transformation and query capabilities are usually proprietary in this approach. Then the EII server is a virtual object database. The Enterprise Data Hub[2] by Journée's (acquired by Initiate Systems) is an example of this approach.

Purely XML-oriented approaches use XML as the logical data model. All data sources are represented as XML document collection and XQuery serves as the language of transformation as well as the query language. The EII server is a virtual XML database. As examples in this category we can mention Ipedo's XIP[3] and Liquid Data for WebLogic (BEA Systems, 2004). In XIP it is possible to query not only collections of XML documents, which is the best known use of XQuery, but also relational databases, web services, common data formats like CSV and fixed length formats. In addition, the Ipedo XQuery engine also allows users to create custom data sources and make them available to XQuery developers. In combination with the distributed SQL query engine, also offered in XIP, these capabilities represent one of the most powerful ways for EII.

However, a use of XML can be only a part of EII solution. There are tools, e.g., MetaMatrix (Hauch *et al.*, 2005), providing an integrated environment for modelling different types of data and information systems. In MetaMatrix deferent layers of metadata are created in more domain-specific languages (including XML Schema), i.e., XML is not a target language here. The support for multiple metamodels is ensured by OMG's MOF (Metadata Object Facility) architecture, i.e., relationships among metadata of different layers are expressed by mapping specifying transformations.

[1] `http://www.iwaysoftware.com/products/eii.html`.
[2] `http://www.initiatesystems.com/`.
[3] `http://www.ipedo.com/html/ipedo_xip.html`.

Although all approaches have their advantages and disadvantages, the XML approach is excellent in data modelling and query capabilities, in particular with applications that use data from nonrelational data sources, such as message queues, EJBs, XML documents, and Web services.

A more advanced approach to integration in EIS is enterprise mushups. Remind that a *mashup* is a Web application that combines content from two or more applications to create a new application. The applications can be built on-the-fly to solve a specific business problem. For example, Damia (Altinel *et al.*, 2007) is inspired by the Web 2.0 mashup phenomenon. It consists of (1) a browser-based user-interface that allows for the specification of data mashups as data flow graphs using a set of operators, (2) a server with an execution engine, as well as (3) APIs for searching, debugging, executing and managing mashups.

Web mashups perform integration both at the application level and at the presentation level. Unfortunately, due to very little support both in terms of model and tools, the presentation part of mashups is developed manually today. An interesting approach to component integration at the presentation level is proposed in Yu *et al.* (2007).

However, the mentioned approaches did nothing to address the semantic integration issues – sources can still share XML files whose tags are completely meaningless outside the application. In consequence, almost all EII products in the market are limited in, or totally lack, the capabilities of semantic interoperability and dynamic adaptation upon changes. Approaches to EII based on Semantic Web technologies like, e.g., RDF and OWL (Booth, 2007; Yuan *et al.*, 2006) are in development today.

3. XML Technology – An Overview

The XML is the universal format for structured documents and data on the Web. An XML document consists of nested XML elements starting with root element. Each element can have attributes and text data, in addition to nested subelements. Text data and subelements can be mixed in an element (so-called mixed content). Child text data and child elements are strictly ordered; attributes are not. XML document can contain also entity references, comments, processing instructions, marked (CDATA) sections, and Document Type Declarations (DTD).

Here is a fragment of the contents of `http://www.bn.com/bib.xml`:

```
<book year="2000">
 <title>Data on the Web</title>
 <author>Abiteboul, Serge</author>
 <author>Buneman, Peter</author>
 <author>Suciu, Dan</author>
 <publisher>Morgan Kaufmann Publishers</publisher>
 <price> 39.95</price>
</book>
```

Beside the basic specification XML namespaces (W3C, 2006) provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references.

Data-centric XML documents are documents that are designed to be read by machines. Data-centric documents are characterised by regular structure. XML elements in these documents generally only contain data or other elements, but not both. The order in which sibling elements occur is generally not significant. Examples of data-centric documents are sales orders, flight schedules, scientific data, and stock quotes. This type of XML document is primarily used to exchange data between applications. Humans rarely, if ever, directly access this type of documents.

Document-centric XML documents are designed to be read and maintained by humans. They are characterised by less regular or irregular structure. Elements in these documents often contain both data and other elements. The order in which sibling elements occur is almost always significant. Examples are books, advertisements, email, Web pages, general message traffic, user-generated comments and almost any handwritten XML documents. These XML documents are always maintained by humans, although applications often need access to certain properties of these documents.

The names of the elements and attributes and their order in the hierarchy (among other things) form the XML markup language used for a collection of documents. We talks about an *XML vocabulary*. There are a lot of XML-based standards for representing content in different industries in which vocabulary is typically defined by a group of XML document users. In the financial sector, industry standards include Financial Products Markup Language (FpML)[4] for financial reporting, Extensible Business Reporting Language (XBRL)[5] for general business, and in publishing – DocBook[6], just to name a few. Health Level Seven (HL7) institute[7] endorsed a recommendation for using XML as an alternative syntax for version V3 designed for electronic messages concerning healthcare workflows. There are also XML standards for representing and integrating technical concepts, such Geography Markup Language (GML)[8] for modeling and exchanging geographic information, Systems Biology Markup Language (SBML)[9], and many others.

Finally, in the context of EII XML is already widely used as a messaging format across industries and for document markup in publishing and government. Remind also ebXML (Electronic Business using eXtensible Markup Language)[10], which is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet.

[4] `http://www.fpml.org/index.html`.
[5] `http://www.xbrl.org/Home/`.
[6] `http://www.docbook.org/`.
[7] `http://www.hl7.org/`.
[8] `http://www.opengeospatial.org/standards/gml`.
[9] `http://sbml.org/Main_Page`.
[10] `http://www.ebxml.org/geninfo.htm`.

4. Database Approach to XML

Traditional databases are based on the notions of a *database model* and a *database schema*. Elements, attributes, mixed content, and other features of XML do not give good assumptions for development of a unique model of XML data. For that reason different XML applications use different models of XML data:

- tree-oriented (e.g., DOM, Infoset, XPath data model, PSVI),
- graph-oriented, supporting ID/IDREF attributes (e.g., OEM),
- object-oriented (ODMG 3.0),
- functional approaches (e.g., Pokorný, 2000),
- combinations of structured and IR models (e.g., Manning *et al.*, 2008).

Any access to XML data must be done through an XML data model. Perhaps the most important XML data model is this one used by languages XQuery, XSLT 2.0, and XPath 2.0. This model is richer than usual tree-like representation. In XPath 2.0, e.g., sequences replace node sets from XPath 1.0.

Solutions of many problems with manipulating XML data rely on a query language. We categorize XML queries into two classes: *databases queries* and *Information Retrieval (IR) queries*. Database queries return all query results that precisely match the queries, which is similar to SQL querying in relational databases. On the other hand, IR queries allow "imprecise" or "approximate" query results, which are ranked based on their relevance to the queries. Only the top-ranked results are returned to users (Manning *et al.*, 2008).

A database user needs not only read operations but also update operations such as inserting, deleting and modifying XML data. Despite of efforts to effectively and efficiently update XML data in last years, only one serious proposal is at disposal, i.e., the W3C working group has completed the XQuery Update Facility (W3C, 2008). It supports XML updates at the node level.

One solution how to store XML data is to use conventional databases. It means to map (shred) the XML documents into data structures of the existing DBMSs (*XML-enabled database*). Detaching generic mappings of XML data into universal tables, usual solutions of this problem have the following properties:

- predefined schema is necessary,
- data is in rows and columns with atomic cell values of multiple tables, which cause that joins are necessary for query evaluations and row ordering is done in an explicit way,
- querying mismatch – navigations in XML data are transformed into SQL, full-text operations are also needed,
- scalability problems.

Another possibility is to store XML data into tables generated algorithmically from an XML Schema which is expressed in an appropriate schema language.

A more advanced solution is to develop a DBMS with a native XML storage (*native XML database* or *NXD*), whose advantages include a support of:

- natural nested hierarchies,

- element ordering,
- documents as single objects,
- schema is not necessary,
- XPath and XQuery have a direct implementation,
- better scalability.

NXDs are most useful for storing document-centric data because they preserve document order, processing instructions, comments, CDATA sections, and entity usage, while XML-enabled databases do not.

In addition to XML-enabled databases and native XML databases, there is the idea of a hybrid database. A *hybrid database* is a relational database that is XML-enabled, but also offers native XML capabilities as defined above. It is a database that supports both the relational data model and the XML data model in all its processing and storage mechanisms.

The XML technology relevant to XML databases concerns mainly XML Schema and query languages. Moreover, query languages contribute to possibilities how specify mapping among heterogeneous data. In the next two subsections we will discuss possibilities that both kinds of languages offer. Their choice in EII design can significantly influence the success of EII in practice.

4.1. *Database Schemas and XML*

Any database schema is a syntactic specification a class of valid data structures, i.e., XML documents in our case. In other words, by schema we describe types of XML documents. In principle, two main possibilities are at disposal: DTD and XML Schema language.

XML Schema provides the ability to define an element's type (string, integer, etc.) and much finer constraints (a positive integer, a string starting with an uppercase letter, etc.). DTDs suppose a strict ordering of elements; schemas have a more flexible range of options (elements can be optional as a group, in any order, in strict sequence, etc.). Finally, schemas are written in XML, whereas DTDs have their own syntax. Other significant XML Schema languages include Relax NG (OASIS, 2001) and Schematron (Jelliffe, 2000). Both RELAX NG and Schematron are intended only for validation of instances, i.e., their possibilities for NXDs are rather restricted. On the other hand RELAX-NG is sometimes preferred because it is simpler and more elegant than DTD or XML Schema languages.

There is certain relationship between schemas expressed in these languages and database schemas. As in other DBMSs, an essential part of each schema definition languages is made by *Integrity Constraints* (ICs). They are useful for semantic specification, update consistency control, query optimization. Comparing to SQL in relational databases, possibilities of ICs in XML Schema are rather poor. Some of them are contained in XML Schema type system, also some cardinality restrictions (minOccurs, maxOccurs) are allowed.

Anyhow the DTD is popular for this XML data specification, it carries a lot of disadvantages. Müller and Schwartzbach (2006) mention top eleven reasons for avoiding DTDs:

- it cannot constraint character data,
- specification of attribute values is too limited,
- element and attribute declarations are context insensitive,
- character data cannot be combined with the regular expression,
- the support for modularity, reuse, and evolution is too primitive,
- the normalization features lack content defaults and proper,
- whitespace control,
- structured embedded self-documentation is not possible,
- the ID/IDREF mechanism is too simple,
- it does not itself use an XML syntax,
- no support for namespaces.

XML Schema (see, W3C, 2004a, 2004b) was regarded to overcome these issues. Nevertheless, there are also problems with XML Schema. Müller and Schwartzbach (2006) list among others the following:

- the details are extremely complicated (and the specification is unreadable)
- declarations are (mostly) context insensitive,
- it is impossible to write an XML Schema description of XML Schema,
- with mixed content, character data cannot be constrained,
- unqualified local elements are bad practice,
- cannot require specific root element,
- element defaults cannot contain markup,
- the type system is overly complicated,
- simple type definitions are inflexible.

As the most important seems the specification expressed in XML Schema is too complicated. An interesting point also is a trade-of between theory and practice: real schemas in XML Schema only rarely use advanced constructs of the language. Most of them are structurally equivalent to a DTD.

The specific problem is to design XML Schemas. There are three ways to design XML Schemas. The first, and the most difficult, is to attempt to create the schema directly element by element. This requires knowing in advance what specific elements already go where. The easier solution is to create an instance of the XML document, then use schema extraction tools to generate a schema that is valid for that instance. The last and most database-oriented possibility uses conceptual modelling XML data. The research in this direction is in progress up to now.

Remind that the conventional approach to schema development includes these processes:

- two phases of design (i.e., a vocabulary and a structure),
- schema evolution, and
- schema versioning.

Unfortunately, today's observation of EIS shows that requirements gathering, schema design and upgrade costs are far more than application development costs. A special feature of XML databases is that many XML documents exist whose schema was *not known*

at design time. Thus, many vocabularies are developed without any schema. As a consequence there are XML databases without any schema. This fact belongs among the key reasons for existence of NXDs. Rather loose possibilities of XML Schema development are much more flexible than relational or object-oriented structural definition. A sufficient compromise between completely schemaless and strict schema-oriented approach is to add cheaply and manageably a small amount of structure which provides a more compelling solution.

4.2. *XML Query Languages*

XML query languages serve to querying, extraction, restructuring, integration, and browsing XML data. They include the following demands:

- pattern matching,
- navigation along the structure of XML tags via (regular) path expressions,
- powerful approach to structured data similar to, e.g., SQL,
- querying both data and metadata,
- generating structured answers to queries (new XML data, derived values).

There are a lot of standards in area of XML query languages designed by W3C, namely XPath 1.0 (W3C, 1999b), XPath 2.0 (W3C, 2007b), XQuery 1.0 (W3C, 2007a), XSLT 1.0 (W3C, 1999a), and XSLT 2.0 (W3C, 2007c). In the context of combining relational and XML data, a special role belongs to the language SQL/XML (ISO/IEC, 2008).

XPath 1.0. A query in XPath uses specified paths applied to a document and rich possibilities to specify relationships between tree nodes. For example, the expression

```
/bib//book[publisher="Kluwer Academic"]/author
```

returns the authors of books with a publisher who has a name Kluwer Academic.

As an answer to a query in XPath we obtain a set of nodes in a typical case. XPath 2.0 is a strict (rather large one) subset of XQuery 1.0. The main use of the XPath language is in other XML query languages, namely XQuery and XSLT.

XQuery language. XQuery is a functional language, comprised `FLOWR (FOR-LET-ORDER-WHERE-RETURN)` clauses that can be nested and composed. Namespace declarations can be used in a query expression and user defined functions as well. Even, a new XML document with derived content can be specified with XQuery. For example, by the expression

```
FOR $book IN document("bib.xml")//book
LET $a := $book/author
WHERE contains($book/publisher, "Kluwer Academic")
RETURN
 <book>
 {
 $book/title,
 <count>Number of authors: { count($a) }</count>
 }
 </book>
```

we specify the number of authors for each book published by Kluwer Academic. The `FOR` clause iterates on a sequence and binds a variable to each node, the `LET` clause binds a variable to a sequence as a whole. From this example it is visible that XQuery includes two parts: twig pattern matching, defined by `FLOWR` expression in the case and the result construction, defined by `RETURN`.

There are various scenarios for use of XQuery, e.g., as a transformation language in Web services or a message broker. The language is also appropriate for data integration and approaching large volumes both centralized of distributed textual data. The latter includes, e.g., XML data sources scattered on the Web. Unfortunately, each scenario requires different processing techniques.

XSLT. XSLT is a language of transformations. It is a declarative language for expressing stylesheets, i.e., instructions that help to transform XML data into a rendered format. Such a transformation takes an XML document and a set of rules (templates) as an input and after processing it by an XSLT processor returns a new XML document. A typical architecture of this processing supposes XML data is in memory.

XSLT vs. XQuery. The focus and strength of XQuery seems to be the data-centric queries (regularly structured markup), while XSLT has its advantages in document-centric queries (semi-structured markup).

Interesting questions appear with languages XQuery and XSLT. Although completely different, they have the same computational power (Kepser, 2004). This implies a question why to support two languages? The answer is usability. Clearly, choosing when use each one is not always easy. Implementation of these languages can be native as well as via relational DBMS. A key observation is a lost of scalability in transforming queries in these languages into sequences of SQL expressions.

SQL/XML. Integration of relational and XML data resulted in development of SQL/XML language. SQL/XML allows relational data to be published in an XML form (XPath data model instance) that can then be queried using XQuery. It provides to define table columns of the XML type. There is a facility called `XMLTABLE` that allows an XML data collection to be viewed as through it were ordinary SQL tabular data, i.e., `XMLTABLE` takes in an XPath or XQuery string, and returns the result as a table. `XMLQUERY` is a function that naturally fits in the SQL `SELECT` clause. It takes two arguments, an XML type object and an XQuery string, and returns an XML type object. Combining SQL and XQuery lets us to query data that is stored in both relational tables and XML in a single query.

XML retrieval. As the web-style searching becomes a ubiquitous tool, the need for integrating exact querying (see languages like XQuery, XSLT, SQL/XML) and IR techniques becomes more important. For example, in EIS environment we meet cases in which users are not able to formulate exact queries, but they provide keyword queries and require a ranking of partial results. In the case of XML, relevance scoring becomes more complex because the data required for scoring could resides not directly in an element itself but also in its descendant elements. Unlike database XML queries, there is no commonly agreed standard language for expressing IR queries. For an excellent survey of XML retrieval, see Pal and Mitra (2007).

4.3. *Architectures of XML Databases: Solutions*

A significant role in storing XML documents is whether the documents are data-centric XML documents or document-centric XML documents.

As we have mentioned earlier, one possibility how to store XML data is an XML-enabled database. In this case, whatever strategy is employed, implementation of a XML query language leads always to generating SQL statements. This translation is known to be difficult, and the resulting SQL can be inefficient. Beyond that, complex XQuery queries can even be untranslatable into SQL. Thus, shredding is most feasible if only simple XPath operations are used or if the applications are designed to work directly against the underlying relational schema. For similar reasons XSLT implementation can be based on use of a relational database, which serves as a temporal storage for source and target XML documents (e.g., Kmoch and Pokorný, 2008).

An implementation of NXD is undoubtedly a challenge in the last years both for developers and researches of database systems. In database architectures, NXDs provide a nice example when a DBMS needs a separate engine (see, e.g., Pokorný, 2007). There are three main approaches to NXD implementation today:

- NXD DBMS as a separate engine,
- adding native XML storage to RDBM (e.g., XML Data Synthesis by Oracle),
- hybrid solution (e.g., IBM DB2 9, ORACLE 11g, SQL Server 2008).

An advantage of the last two approaches is the possibility to mix XML with relational data. While critical data is still in a relational format and the data that not fit the relational data model is stored natively in XML.

With the new option of storing and querying XML in a RDBMS, schema designers face to the decision of what portion of their data to persist as XML and what portion as relational data. ReXSA described by Moro *et al.* (2007) is a schema advisor tool that is being prototyped for IBM DB2 9, enabling to propose candidate database schemas given a conceptual model of the enterprise data.

Bourret (2007a) registers[11] 42 NXD products and remarks there is certainly no clear market leader as yet. We list a short list of the most cited NXDs (Table 1).

Adding to those from Section 4, NXDs may include these features:

- internal identity management systems and integration with external identity management systems like LDAP (lightweight directory access protocol),
- seamless, schema-independent persistence/caching of Java Web service messages, XML, SOAP, and WSDL (Web Services Description Language),
- built-in support for security standards such as Security Assertion Markup Language (SAML), Web Services Security (WSS), and XML Encryption,
- built-in support for workflow management based on Business Process Execution Language (BPEL),
- seamless persistence of unstructured content,
- interactive and intuitive graphical environment,

[11] In March 2007.

Table 1
A list of the most cited NXDs

Tamino	Developer: Software AG (Germany) URL: http://www.softwareag.com/tamino/ Query languages: X-Query (now XQuery)
XHive/DB	Developer: X-Hive Corporation URL: http://www.x-hive.com/products/db/ Query languages: XQuery, XPath, XML Schema
XIndice	Developer: The Apache Software Foundation URL: http://xml.apache.org/xindice/ Query languages: XPath, XUpdate
Berkeley DB XML	Developer: Sleepycat Software URL: http://www.oracle.com/ database/berkeley-db/xml/index.html Query languages: XQuery
MarcLogic Server	Developer: dbXML Group URL: http://www.marklogic.com/ Query languages: XQuery
eXist	Developer: Wolfgang Maier URL: http://exist.sourceforge.net/ Query languages: XQuery/XPath 2.0

- seamless integration with external JDBC sources with ability to read, query, insert, update, and delete all within an XA-compliant[12] transaction,
- seamless integration with HTTP and SOAP sources,
- transactions with all available data sources using XQuery,
- standards compliance by enforcing schema validation and data aggregation mapped to a required schema,
- backup, restore, and replicate capabilities.

5. EII through XML Technology

To store XML data in a database should mean to manage large numbers of XML documents in a more effective way. Although this idea looks attractively there is also scepticism from the side of XML database developers. For example, M. Kay (Software AG) says: I generally argue that XML is designed primarily for information interchange, and that the requirement for storage is secondary.

[12] An XA compliant driver is a driver that can participate in an XA compliant transaction as defined by the X/Open.

A motivation for maintaining XML data in databases has roots in application demands, in particular to ensure a better work with content in enterprises. With an XML database one can, e.g., process external data (Web pages, other text databases, structured data), resolve tasks of e-commerce (lists of products, personalized views of these lists, orders, invoices in e-commerce, e-brokering), and support integration of heterogeneous information sources. A typical example of the latter is an integrated processing data from Web pages and from tables of a relational database. There are XML database vendors who market their platforms as EII solutions (e.g., Software AG, IBM, Ipedo).

A great debate concerns question when to use NXDs and even what is the purpose of NXDs. Bourret (2007b) distinguishes two types of uses of NXD. Among primary uses he classes storing and querying document-centric XML data, data integration, and processing semi-structured data. Semi-structured data has some structure, data is usually self-describing (usually schema-less). Examples of semi-structured data include data about schema evolution, biological data, metadata, health data, catalogues, emails, and (possibly surprisingly) economical data. Secondary uses of NXD include long-running transactions, handling rapidly evolving schemas, working with very large documents, querying hierarchical data, and running Web sites.

Since storing and querying XML data as well as data integration are crucial for EII we focus on these kinds of NXDs uses in detail.

5.1. *Storing and Querying Document-Centric XML Data*

A good example is provided by content management. It is well-known that reuse represents an important way for companies to extend the value of their investment in content. According to the study by ZapThink (2003), producers of content spend over 60% of their time locating, formatting, and structuring content and just 40% for creating the content.

Moreover, the great bulk of content within an enterprise falls within the unstructured domain and stand-alone relational DBMSs are not well prepared for management such content. XML offered a robust technology that became a background of *content management systems* (CMS). In the approach based on XML, the software architecture is called also *XML content server*. Such systems provide users tools for automatic conversion and distribution of native content via the Web. As XML separates formatting data from XML content, a new trend is to build CMS on the top of NXD. As a consequence of using XML the distinction between structured and unstructured information may now be blurring.

Two advantages with XML are significant in the context of CMS:

- adoption of XML by the IT industry at large,
- availability of simpler user interfaces for authoring content in XML.

By SYS-CON Media Inc. (2008), the following XML features are essential in the context of CMSs:

- *Content contribution and conversion*. CMS contains information in a native format, e.g., text document, image, spreadsheet, etc. Storing content in XML enables its various transformations into a variety of formats, such as HTML, for reuse by multiple applications.

- *Content access and exchange*. XML content can be easily merged with other sources and represented in a unified way in content management repository.
- *Content formatting and presentation*. In CMS XML separates formatting data from XML content. A separation of content and presentation allows different formatting to be applied to the same content in different situation using XML stylesheets.
- *Content storage*. CMSs based o the top of NXD provide a number of benefits comparing to solutions integrating content with relational databases. For example, XML content stored in an XML database cane be more easily searched with the help of query languages like XQuery or XPath.
- *Content personalization*. Based on user profiles and type of device, CMS can deal with the content accommodated by an associate XSL stylesheet. Such tailored content is then delivered to the user.
- *Content management Web services*. XML plays an important role in Web services. Most of CMSs use Web services to share and deliver data and specific content management features in the Internet.

It seems, that XML databases have separated into three subcategories, and, consequently, with three groups of vendors. One group has focused on managing XML content or documents (e.g., Mark Logic). For example, MarkLogic Server provides a platform for CMS combining traditional DBMS based on XML with full-text searching. The other two categories are related with EISs.

5.2. *Data Integration*

XML database can provide a middle tier Operational Data Store (ODS) platform. In the third category of uses, XML database focuses on managing persistent data on a middle tier for data integration applications, in particular EII applications (e.g., Ipedo).

- *Operational data store*. A middle tier ODS can provide the necessary infrastructure for managing enterprise data and bringing it closer to the consuming business application, while simultaneously reducing the burden on backend systems of record. XML databases are an ideal technology to serve as an ODS because of their ability to maintain schemas and to bind heterogeneous data sources. Furthermore, XML databases' support for XA-compliant transactions make them an ideal ODS and EII technology that enables both read and write capabilities across heterogeneous systems.
- *Enterprise information integration*. XML databases enable EII by providing a platform for querying across heterogeneous data sources, resulting in view of all common entities spread across enterprise systems or services. EII provides huge benefits to business users. Typically, CMSs can become a source of integration in EII. Although relatively new, most current EII approaches are still based on similar principles of loosely-coupled federated systems. Moreover, a key issue, i.e., resolving differences in schemas and integrate them into one central schema, is often not requited in today's EII applications. In such systems data is managed as schemaless, eliminating the need for schema management and database administration. Obviously, finding information is a prerequisite to integrating it.

5.3. *Towards EII and Web Services Integration*

Web services create huge amounts of new data, specifically the exchange of data-rich XML messages. These messages contain important information that many organizations will want and need to store, access, query, audit, analyze, and repurpose. It is nearly impossible to persist all of these messages in a relational database because of the inflexible data model they impose. It is necessary to know what type of data the message will contain and set up relational tables to store it. Additionally, one will have to write code that knows, for every message type, how to take the incoming message, shred it, and populate the tables.

Here it is possible to use NXD as a "glue" to connect existing enterprise systems. For example, in SOAP XML-based object serialization format can be used to perform asynchronous messaging and RPC between non-XML applications. Although messages are probably data-centric, their natural format is XML. It makes sense to build a message queue on a NXD, particularly in cases when EII is *event-driven* rather than *query-driven*. Then data changes, for example, could be accumulated in a message queue and an EII query scheduled to run at periodic intervals to read the data from the queue and update a data store with the changes. We obtain XML-specific capabilities and, consequently, better scalability as the volume and complexity of e-business transactions increases. XML databases are particularly useful for handling new message types or evolving message structures. Storing message content in a native XML database reduces the development time and cost at least 50% by elimi nating the need to define object-to-relational mapping.

The immediate popularity of XSLT accelerated the EAI development, and some use of XSLT is probably a requirement in all EII solutions today. Any exchange of XML information is namely going to involve a combination of mapping of information objects, and in most cases these will involve structural transformations to account for different contexts, i.e., uses of the information. There is simply no reason to use anything but XSL for this.

Uses for a native XML database include:

- providing a unified master data-access layer across the enterprise,
- validating, persisting, querying, and repurposing XML,
- becoming XML-standards compliant,
- aggregating content from a variety of systems JDBC, HTTP, file system, Web services),
- serving as an enterprise data cache and operational data store to improve data-access response times and relieve burden on backend systems,
- supporting an enterprise data bus solution,
- intelligent tools to repurpose the data (using XSLT and XQuery).

A trend is apparent now. The ability to efficiently store and access XML and relational data types in one system represents a key point of differentiation among enterprise database vendors. It also allows enterprise developers to build data-driven applications using XML data types. Open source RDBMS from companies such as PostgreSQL currently also supports this hybrid XML-relational data-storage capability.

6. Open Questions for R&D

Today's demands on XML processing applicable not only on EII, but also in general, include:

- *New query capabilities based on XML retrieval*. This means a retrieval not only via key words; more generally, user defined functions are required, like automatic abstract generating, techniques based on the notion of document similarity or proximity between terms. In other words, queries as "something about X" or "something like X" are needed.
- *Conceptual modelling XML data*. As XML gets near to the core of applications, a modelling of XML data should become an inseparable part of modelling of application data on the conceptual level. Today, structure of XML data is designed usually directly, without the conceptual schema. This makes more difficult, e.g., modelling hierarchies like it is used in ER modelling. With XML conceptual modelling also automatic transformations to XML Schema is easier and more accurate. The research in this area is represented, e.g., by Nečaský (2007).
- *Schema extraction given a body of XML data*. Since schema definitions are not obligatory in XML documents, there is often a need to extract the schematic information from XML documents. The extracted schema should, one side, tightly represent the data, and be concise and compact, on the other side. As the two requirements essentially contradict each other, finding an optimal trade-off is a difficult and challenging task. Today's attempts do not go far than to DTD inference (see, e.g., Garofalakis *et al.*, 2000).
- *Language for expressing complex ICs*. XML data, just like traditional databases, may be specified by both type constraints and ICs, such as keys, foreign keys, and functional dependencies, which depend primarily on the equality of data values. Due to the hierarchical nature of XML data generalizing relational ICs to XML ICs is not trivial. Remind that languages like DTD and XML Schema offer only very limited possibilities for ICs formulation. An excellent overview of these issues is in Fan (2005).
- *Implementation issues*. Finally (never ending) issues concern representation of XML data in external memory and its indexing. Such a representation influences effectiveness of XML query languages as well. Improvements of XML-enabled approach can be also achieved accessing XML content via object-relational DBMSs.

7. Conclusions

We have shown that EII is a broad notion that raises more questions than it answers. In the survey (Zhou *et al.*, 2006) the authors address four challenges of EII including scalability, horizontal vs. vertical integration, central integration, and semantics. For example, with an increasing number of sources, the scale-up efficiency decreases. Integration is mostly horizontal than vertical in the most EII systems and the only vertical part is their

centralized administration. Really significant challenge is information sharing, which requires considering more semantics in EII. As we mentioned in the paper, techniques of the Semantic Web can contribute to this problem. Despite the success of the first XML applications in EII it seems that it can still help significantly to respond to these challenges.

Acknowledgements

This research has been supported by the grant of GACR No. GA201/09/0990.

References

Altinel, M., Brown, P., Cline, S., Kartha, R., Louie, E., Markl, V., Mau, L., Ng, Y.-H., Simmen, D., Singh, A. (2007). Damia – A data mashup fabric for intranet applications. In: *VLDB '07*, Vienna, Austria, pp. 1370–1373.

Antoniou, G., van Harmelen, F. (2004). *A Semantic Web Primer*. The MIT Press (April 1, 2004).

BEA Systems (2004). Liquid data for WebLogic: Integrating enterprise data and services. In: *Proc. of SIGMOD 2004*, Paris, France, pp. 917–918.

Booth, D. (2007). RDF and SOA. In: *W3C "Web of Services" Workshop*. `http://dbooth.org/2007/rdf-and-soa/rdf-and-soa-paper.htm`.

Bourret, R. (2007a). *XML Database Products*. `http://www.rpbourret.com/xml/XMLDatabaseProds.htm`.

Bourret, R. (2007b). *Going Native: Use Cases for Native XML Databases*. `http://www.rpbourret.com/xml/UseCases.htm`.

Fan, W. (2005). XML constraints: Specification, analysis, and applications. In: *Proc. of DEXA'05 Conf.*, Copenhagen, Denmark, pp. 805–809.

Fremantle, P., Weerawarana, S., Khalaf, R. (2002). Enterprise services. *Communications of the ACM*, 45(10), 77–82.

Garofalakis, M., Gionis, A., Rastogi, R., Seshadri, S., Shim, K. (2000). Xtract: A system for extracting document type descriptors from XML documents. In: *Proc. of SIGMOD Conf.*, Dallas, Texas, USA, pp. 165–176.

Gilbane, F. (2004). What is Enterprise Information Integration (EII)? *The Gilbane Report*, Vol. 12(6), Bluebill Advisors, Inc., 1993–2005.

Hauch, R., Miller, A., Cardwell, R. (2005). Information intelligence: Metadata for information discovery, access, and integration. In: *Proc. of SIGMOD Conf.*, Baltimore, Maryland, USA, pp. 793–798.

Halevy, A.Y., Rajaraman, A., Ordille, J. (2006). Data integration: The teenage years. In: *Proc. of VLDB '06*, Seoul, Korea, pp. 9–16.

Halevy, A.Y., Ashish, N., Bitton, D., Carey, M.J., Draper, D., Pollock, J., Rosenthal, A., Sikka, V. (2005). Enterprise information integration: Successes, challenges and controversies. In: *Proc. of SIGMOD Conf.*, Baltimore, Maryland, USA, pp. 778–787.

ISO/IEC 9075-14 (2008). *Information Technology – Database Languages – SQL, Part 14, XML-Related Specifications (SQL/XML)*.

Jelliffe, R. (2000). *The Schematron: An XML Structure Validation Language Using Patterns in Trees*. `http://xml.ascc.net/resource/schematron/`.

Lu, E.J.-L., Wu, B.-Ch., P Chuang, P.-Y. (2006). An empirical study of XML data management in business information systems. *Journal of Systems and Software*, 79(7), 984–1000.

Kepser, S. (2004). A simple proof of the Turing-completeness of XSLT and XQuery. In: T. Usdin (Ed.), *Extreme Markup Languages*. IDEAlliance. `http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Kepser01/EML2004Kepser01.html`.

Kmoch, O., Pokorny, J. (2008). XSLT Implementation in a relational environment. In: *Proc. of the IADIS Multi-Conference on Computer Science and Information Systems* – Subconference *Informatics*, Amsterdam, The Netherlands, pp. 91–98.

Manning, Ch.D., Raghavan, P., Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Mattos, N.M. (2003) Integrating information for on demand computing. In: *Proc. of VLDB'03*, Berlin, Germany, pp. 8–14.

Moro, M.M., Lim, L., Chang, Y.-C. (2007). Schema advisor for hybrid relational-XML DBMS. In: *SIGMOD'07*, Beijing, China, pp. 959–970.

Müller, A., Schwartzbach, M.I. (2006). *An Introduction to XML and Web Technologies*. Addison-Wesley.

Nečaský, M. (2007). XSEM – A conceptual model for XML. In: Roddick, J.F., Annika, H. (Eds.), *Proc. Fourth Asia–Pacific Conference on Conceptual Modelling* (*CRPIT*), Ballarat, Australia, Vol. 67. Australian Computer Society, pp. 37–48.

OASIS (2001). *RELAX NG Specification*. `http://www.oasis-open.org/committees/relax-ng/spec-20011203.html`.

Pal, S., Mitra, M. (2007). XML Retrieval: A Survey, Technical Report, CVPR. TR/ISI/CVPR/IR07-01.

Pokorný, J. (2000). XML functionally. In: Desai, B.C., Kioki, Y., Toyama, M. (Eds.), *Proc. of IDEAS2000*. IEEE Comp. Society, pp. 266–274.

Pokorný, J. (2007). Database architectures: Current trends and their relationships to requirements of practice. In: *Advances in Information Systems Development*: *New Methods and Practice for the Networked Society*. Springer-Verlag, pp. 269–279.

Smik, R., Parikh, A., Ramachandran, A. (2004). Use XML databases to empower Java Web services. JavaWorld.com, 12/06/04.

SYS-CON Media Inc. (2008). The role in XML in content management. *XML Journal*.

Tailor, J.T. (2004). *Enterprise Information Integration: A New Definition*. Integration Consortium, DM Review Online, September 2, 2004.

W3C (1999a). *XSL Transformations (XSLT)*, Version 1.0. W3C recommendation.

W3C (1999b). *XML Path Language (XPath)*, Version 1.0. W3C recommendation. `http://www.w3.org/TR/xpath`.

W3C (2004a). *XML Schema, Part 1, Structures* (Second Edition). W3C recommendation. `http://www.w3.org/TR/xmlschema-1/`.

W3C (2004b). *XML Schema, Part 2, Datatypes* (Second Edition). W3C recommendation. `http://www.w3.org/TR/xmlschema-2/`.

W3C (2004c). *Extensible Markup Language (XML) 1.1*. (Second Edition). W3C recommendation. `http://www.w3.org/TR/xml11/`.

W3C (2005). *XQuery 1.0, An XML Query Language*, W3C Working Draft, 04 April 2005. `http://www.w3.org/TR/xquery/`.

W3C (2006). *Namespaces in XML 1.0* (Second Edition). W3C recommendation. `http://www.w3.org/TR/xml-names/`.

W3C (2007a). *XQuery 1.0, An XML Query Language*. W3C recommendation. `http://www.w3.org/TR/xquery/`.

W3C (2007b). *XML Path Language (XPath)*, Version 2.0. W3C recommendation. `http://www.w3.org/TR/xpath20/`.

W3C (2007c). *XSL Transformations (XSLT)*, Version 2.0. W3C recommendation. `http://www.w3.org/TR/xslt20/`.

W3C (2008). *XQuery Update Facility 1.0*. W3C candidate recommendation. `http://www.w3.org/TR/xqupdate/`.

Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M. (2007). A framework for rapid integration of presentation components. In: *WWW 2007*, Banff, Alberta, Canada, pp. 923–982.

Yuan, J., Bahrami, A., Wang, Ch., Murray, M., Hunt, A. (2006). A semantic information integration tool suite. In: *Proc. of VLDB'06*, Seoul, Korea, pp. 1171–1174.

ZapThink (2003). Market for XML-enabled content lifecycle solutions to exceed $ 11.6 billion by 2008; XML key to solving critical content management problem: Content reuse. Business Wire.

Zhou, J. Wang, M., Zhao, H. (2006). Enterprise information integration: State of the art and technical challenges. In: *Proc. of PROLAMAT, IFIP TC5 International Conference*, pp. 847–852.

J. Pokorný, prof. RNDr., CSc. received the PhD degree in theoretical cybernetics from the Charles University, Prague, Czechoslovakia, in 1984. Currently he is a full professor of computer science the Faculty of Mathematics and Physics at Charles University and the member of its Department of Software Engineering whose head he was from 1994 to 2006. As the deputy-dean and vice-dean for research and international affairs he works from 2008. He is also a visiting professor at Faculty of Electrical Engineering of the Czech Technical University, Prague. J. Pokorny has published more than 250 papers and books on data modelling, relational databases, query languages, file organization, and XML technology. His research interests include also information retrieval, semistructured data, and indexing methods. He is a member of ACM and IEEE.

XML organizacijų informacinėse sistemose

Jaroslav POKORNÝ

XML formatas plačiai paplito kaip duomenų aprašymo būdas bei įsitvirtino kaip žinučių, kuriomis apsikeičia šalys, formatas. Paskutiniais metais duomenų integravimo technikoms tai padarė nemažą įtaką. Visa eilė sukurtų XML kalbų, įvairūs metodai bei įrankiai prisidėjo prie taip vadinamos XML technologijos atsiradimo. Organizacijos informacijos integracija (EII) reikalauja tikslaus ir pilno skirtingų duomenų šaltinių, informacijos vartotojų poreikių supratimo, bei gebėjimo visa tai susieti su organizacijos verslo sąvokomis. Bet kokia integracija vyksta organizacijos informacinės sistemos (EIS) kontekste. Šiame straipsnyje mes analizuojame įvairius būdus, kaip galima atlikti organizacijos informacijos integraciją, pateikiame galimas jos architektūras bei sąsajas su organizacijos sistemų integracija (EAI). Mes pasiūlome organizacijos informacijos integracijos bazinių savybių bei iššūkių rinkinį ir pagrindžiame, kodėl XML technologija padeda surasti pakankamai galingas priemones organizacijos informacijos integracijos įgalinimui. Konkrečiai grynųjų XML duomenų bazių atsiradimas pateikia universalų sprendimą, įgalinantį konstruoti organizacijos informacijos integracijos instrumentus. Straipsnyje mes pateikiame kai kurias XML technologijos savybes – didžiąja dalimi susijusias su grynosiomis XML duomenų bazėmis, ir parodome, kuo jos naudingos organizacijos informacijos integracijai.