XML-based EIS – A Meta Schema for Mapping XML Schema to Relational DBS

Elisabeth Kapsammer
Information Systems Group (IFS), Department of Bioinformatics
Johannes Kepler University Linz, Austria
ek@ifs.uni-linz.ac.at
http://www.ifs.uni-linz.ac.at/ifs/staff/kapsammer/kapsammer.html

Abstract. Enterprise information systems (EIS) often employ relational database systems to store their content. At the same time, XML constitutes the dominant standard for data exchange as well as for the hypertext level of web-enabled EIS. Thus, the integration of XML with relational database systems to allow the storage, retrieval, and update of XML documents is of paramount importance in the context of EIS. Data model heterogeneity and schema heterogeneity, however, make this a challenging task. This paper proposes X-Ray, a generic approach facilitating the composition of XML documents out of a relational database system as well as the decomposition of XML documents to store them within a relational database system in a fully transparent way. This is achieved by means of a generic meta schema that stores all relevant information for the composition and decomposition process. This meta schema covers both, the concepts of DTDs and XML Schema concepts.

1 Introduction

Enterprise information systems (EIS) typically employ databases (DB) to store their content. At the same time, the Extensible Markup Language (XML) 16 is fast emerging as the dominant standard for data exchange between different EIS 19, 20. Furthermore, regarding the realization of EIS based on Web technology, where databases constitute the *content level*, XML has become the first choice for representing data at the *hypertext level* of a web site, i.e., the logical composition of web pages and the navigation among them 14. Because of the great importance of XML and database systems (DBS) in the context of EIS, the integration of them with respect to storage, retrieval, and update is a major need. Such integration also enables the transformation of legacy data to various data formats via XML.

Please use the following format when citing this chapter:

Kapsammer, E., 2006, in International Federation for Information Processing, Volume 205, Research and Practical Issues of Enterprise Information Systems, eds. Tjoa, A.M., Xu, L., Chaudhry, S., (Boston:Springer), pp.73-84.

Regarding the DBS's data model underlying an XML-based EIS used for the integration we concentrate on the *relational data model*, which is especially motivated by the fact that in EIS currently, a significant amount of data is stored in pre-existing relational databases (RDBS) and will continue to be used by existing EIS in the future 5. There is an increasing demand to publish existing relational data as XML according to existing standardized XML schemata in terms of a document type definition (DTD) 16 or the more powerful XML Schema language 17* or, vice versa, for storing XML documents in existing DB incorporated by EIS.

Concerning the integration with RDBS one can distinguish three alternatives as well as combinations thereof. XML documents as a whole can be stored within a *single database attribute*, XML documents may be *decomposed* in some way ("shredding"), e.g., into a *graph structure* and stored into appropriate database tables, and finally, the *structure of XML documents may be mapped* to a corresponding relational schema wherein XML documents are stored according to the mapping (cf., e.g., 3). Only the last approach allows reusing existing relational schemata and thus is further investigated in this paper. One major challenge of this schema-to-schema mapping approach is the existence of *data model heterogeneity* and *schema heterogeneity*. In this respect, an important demand on a particular integration solution is that the *autonomy* of both the XML schema specification and the relational schema should be preserved in that neither of them has to be changed.

We have already proposed a generic approach that solves such an integration need prevalent in EIS in 7. X-Ray facilitates the composition of XML documents out of a relational database system as well as the decomposition of XML documents to store them within a relational database system in a fully transparent way. The key mechanism for the genericity of X-Ray is constituted by a meta schema that stores all relevant information to map DTDs and relational schemata. Since the adoption of XML Schema as schema specification standard for XML documents by the W3C in addition to DTDs, more and more XML documents rely on XML Schema. Therefore, this paper discusses the enhancement of X-Ray to support XML Schema, primarily focusing on the design of the meta schema.

The remainder of the paper is organized as follows. Section 2 discusses several issues of data model heterogeneity by comparing concepts of RDBS and XML schema specification languages. Section 3 gives an overview of X-Ray in terms of its design goals and architecture. Section 4 gives an insight into the meta schema, focusing on the XML Schema part. In Section 5 related work is compared to X-Ray. Finally, Section 6 concludes the paper with an outlook to future work.

2 A Tour on DTDs, XML Schema, and RDBS

Analyzing and understanding the different kinds of heterogeneities between DTDs, XML Schema, and RDBS constitutes the prerequisite for designing an appropriate meta schema for mediation purposes. Therefore, this section discusses the most crucial heterogeneity aspects. For a more detailed discussion it is referred to 9.

* Note, that in the following, we use a capital letter to denote the XML *Schema* standard and a small letter to depict any XML *schema*.

2.1 DTD versus XML Schema

The main differences between DTDs and XML Schema concern the issues of syntax, namespaces, integration mechanisms, data types, identification, and references. First of all, concerning the *syntax*, XML Schema is based on XML itself instead of using a proprietary syntax as done by DTDs. Second, opposed to DTDs, XML Schema supports *namespaces* and provides improved mechanisms to integrate different schemata. Third, concerning *typing mechanisms* XML Schema distinguishes between elements and types, whereas DTDs allow specifying element types, only. XML Schema supports not only the definition of types but also elements with same name, but different structure. In addition, DTDs support a few data types, only, whereas XML Schema provides various data types and user defined type hierarchies. Fourth, DTDs offer limited concepts to realize unique *identifiers* and *references*, in form of the data types ID and IDREF(S). These data types may be applied to XML attributes; only, their values are limited to so called XML names. Thus, neither numbers nor composite keys are allowed. The scope of uniqueness of an ID value and the references established by IDREF(S) comprise the whole XML document, instead of parts thereof. In contrast, XML Schema provides a powerful key and keyref concept comparable to the key concept well known from relational DBS.

2.2 XML versus RDBS

Similar to heterogeneities between DTDs and XML Schema, there are also several kinds of heterogeneities with respect to RDBS. These comprise the relevance of a schema, structuring and typing mechanisms, storage of values, uniqueness of names, identification, relationships, and order. First, concerning the *schema relevance* aspect XML schemata are optional, can be designed a posteriori, and are implicitly part of each XML document in form of tags. In RDBS, schemata are mandatory, have to be specified a priory, and are not replicated as part of the content. Second, regarding *structuring* and *typing mechanisms*, whereas RDBS allow a flat structure made up of relations comprising attributes, only, XML supports an arbitrary nested structure, consisting of elements of certain element types, and attributes. Element types can be categorized along two dimensions. The first dimension depicts whether the element type *contains an atomic domain* whereas the second dimension denotes whether the element type *contains a composite domain*. This distinction results in four different kinds of element types, i.e., atomic, composite with element content or mixed content, respectively, and empty. Third, looking at the *storage of values*, in RDBS values are stored within attributes, only, whereas XML allows storing values within both, elements and attributes. Fourth, regarding *uniqueness of names*, the name of a relation is required to be unique within the whole relational schema, similar to the name of an XML element type being unique throughout the DTD. XML Schema is more flexible in this respect since the name of an XML element type has to be unique within a so-called *symbol space,* only. Moreover, by means of so called *namespaces* 15, XML allows element types having the same name by using different namespace prefixes. The name of an XML attribute has to be unique within its element type, again similar to an RDBS attribute's name which has to be unique within its relation. Fifth, concerning *identification* and *relationships*, RDBS provide the well-known key and foreign key concepts that realize uniqueness of possibly composed values within a single relation and typed references. Finally, elaborating on the significance of *order*, in RDBS relations and tuples are not ordered, whereas in XML, element types and elements occur in a certain order.

3 X-Ray at a Glance

This Section gives an overview of X-Ray by focusing on the underlying design goals and by introducing its architecture.

3.1 Design Goals

The development of X-Ray as a core middleware technology for EIS was driven by the following design goals: In order to allow the integration of *existing XML schemata* and *existing RDBS schemata*, which shall *remain autonomous*, X-Ray supports a *loose coupling* by defining *explicit mapping knowledge*. To achieve *mapping transparency* and *reduce maintenance effort*, X-Ray stores the mapping knowledge *reified within a DB*. In order to support *multiple schemata* at both sides, X-Ray allows mapping a certain schema to multiple schemata at the other side. To enhance its *universal applicability* X-Ray supports both *publishing* as well as *storage* of XML documents. To establish *schema transparency*, X-Ray provides a *virtual XML view* over the RDBS, being the target of a query when accessing X-Ray. Finally, to provide *homogeneous access*, X-Ray realizes an *XML-centric* solution by using XQuery as query language to access XML data stored within the RDBS.

3.2 Architecture

The architecture of X-Ray consists of three main parts: the generic *meta schema*, the *mapping knowledge editor*, and the *composer/decomposer* component (cf. Fig. 1).

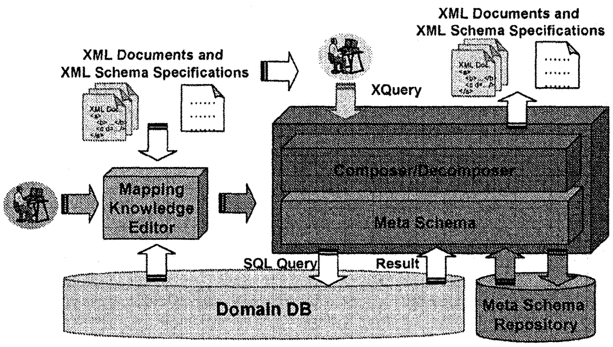
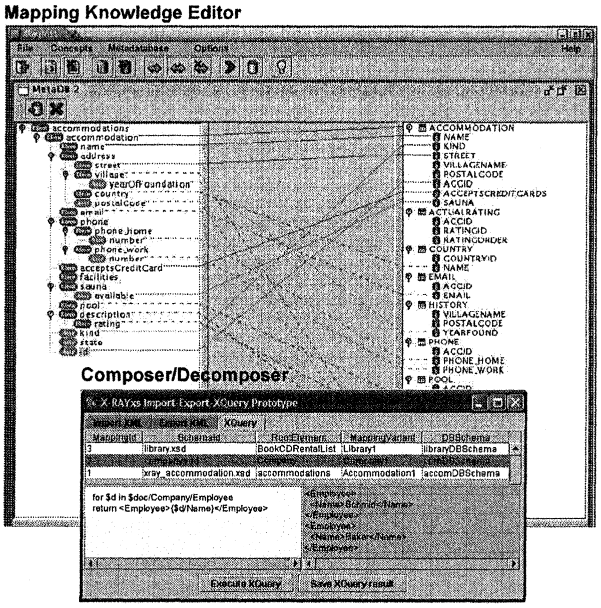


Fig. 1. Architecture of X-Ray

Before X-Ray can be used for storing and retrieving XML documents, the mapping knowledge required for mapping a certain XML schema to a certain relational schema has to be specified in an initialization phase. To support this task, the X-Ray architecture provides a *mapping knowledge editor*. On the basis of a database schema and an XML schema, the user may interactively specify the required mappings, guided by proposed mapping patterns (cf. 9). As soon as the system is initialized with the mapping knowledge which is stored within the meta schema repository, the user is able to transparently issue queries using XQuery 18 against a virtual XML view. Utilizing the mapping knowledge, the query is decomposed into corresponding SQL queries on the relational database. The result is used to compose XML documents out of flat relational data. The *composer/decomposer* component serves for storing and retrieving XML documents and therefore performs all necessary transformations based on the mapping knowledge. A first prototype of X-Ray, implemented on basis of Java and Oracle9i is already operational. Fig. 2 shows the *mapping knowledge editor* and the interface of the *composer/decomposer* component responsible for storing and retrieving data.



MappingId	SchemaId	RootElement	MappingVariant	DBSchema
3	library.xsd	BookCDRentalList	Library1	libraryDBSchema
[illegible]	[illegible]	Company	[illegible]	[illegible]
1	xray_accommodation.xsd	accommodations	Accommodation1	accomDBSchema

```
for $d in $doc/Company/Employee
return <Employee>{$d/Name}</Employee>
```

```
<Employee>
  <Name>Schmid</Name>
</Employee>
<Employee>
  <Name>Baker</Name>
</Employee>
```

Fig. 2. Mapping Knowledge Editor and Composer/Decomposer of X-Ray

The mapping knowledge editor displays on the left hand side a certain schema for XML documents by means of graphically representing an XML Schema, whereas the right hand side shows a relational schema comprising relations together with their attributes. Mappings between these schemata are depicted by lines connecting them in a proper way. The *composer/decomposer* shows the choice of a mapping between two schemata and the selection of a particular root element. With respect to this choice, an XQuery can be issued and the corresponding result is displayed.

4 Meta Schema of X-Ray

The insights gained previously concerning data model heterogeneity and the design goals provide the basis for the meta schema. The main task of this meta schema is to mediate between heterogeneous concepts at the XML side and the relational side. Thus, it provides the basis for EIS, automatically composing XML data out of the RDB when requested and decompose them when they have to be stored.

4.1 Overall Structure of the Meta Schema

Basically, the meta schema consists of three parts describing the relevant meta knowledge (cf. the UML package diagram in Fig. 3). The `DBSchema` part is responsible for storing information about the relational schemata that shall be mapped to XML schemata. It contains information about relations, database attributes, relationships, and joins. Analogously, the `XMLSchema` part stores information about XML documents as specified by means of DTDs and XML Schemata, respectively. Finally, the `XMLDBSchemaMapping` component stores the user-defined mapping knowledge between `DBSchema` and `XMLSchema`. The goal of `XMLDBSchemaMapping` is to bridge both data model heterogeneity and schema heterogeneity in order to support a proper mapping.

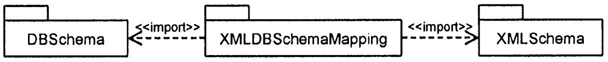


Fig. 3. Basic Parts of the X-Ray Meta Schema

In X-Ray, a database schema is not limited to be mapped to a single XML schema but may be mapped to several XML schemata and vice versa. Concerning the storage of the meta knowledge, X-Ray comprises both a relational representation of the meta schema stored within a relational database and an object-oriented representation for main memory mapping. The latter is being initialized with the content of the relational meta schema at the beginning of an X-Ray session, herewith allowing an efficient composition and decomposition of XML documents at runtime. The object-oriented representation is depicted in the following in terms of UML class diagrams. Thereby, it is focused on the `XMLSchema` part, whereby, for representation convenience, we concentrate on several classes and relationships, only. For details about the `DBSchema` and the `XMLDBSchemaMapping` part it is referred to 9.

4.2 XMLSchema Part of the Meta Schema

As already mentioned, the `XMLSchema` part stores information about both, DTDs and XML Schemata. Note, that X-Ray currently supports not the full range of concepts offered by the XML Schema standard. The selection of concepts supported by X-Ray in this first stage of development was driven by Pareto's 80/20 rule, since practice 11 has shown that there is a bundle of core XML Schema concepts which is used for most problems at hand. For discussion of those concepts not currently supported by our meta schema it is referred to Section 6. To reduce complexity, `XMLSchema` is composed into four core packages, namely `XS_SchemaComposition`, `XS_Types`, and two sub packages (`XS_SimpleUDTypes` and `XS_ComplexUDTypes`), `DocumentContent`, and `ElementContent` (cf. Fig. 4).

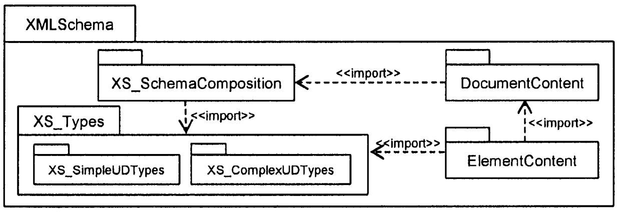


Fig. 4. Core Packages of XMLSchema

Note, packages prefixed with XS_ store knowledge specific to information retained in XML Schemata and will be further discussed in this paper. DocumentContent and ElementContent describe information relevant for both, DTDs and XML Schema and are described in 9, discussing a meta schema for DTDs. The packages XS_SchemaComposition, XS_Types, XS_SimpleUDTypes, and XS_ComplexUDTypes are described in the following in more detail.

The package XS_SchemaComposition allows storing information about XML Schema documents, their associated namespaces as well as the composition structure of different XML Schema documents (cf. Fig. 5). A namespace may be assigned to an XML Schema by the hasTargetNS relationship that allows associating a prefix with the namespace via an association class. Concerning the composition structure, three different kinds of relationships between documents can be distinguished, covered in the meta schema by using recursive associations, namely include to incorporate documents of the same namespace or without namespace, redefine, a special form of include that enables to change certain specifications, and import, allowing to combine XML Schema documents of different namespaces. Whether namespaces of the integrated schemata may be the same or not is determined by appropriate constraints. The fact that redefine is a special form of include is expressed by means of inheritance between the respective associations, whereby the specialized relationship redefines is extended by the association class Redefinition, to specify further details. Similar to hasTargetNS the relationship imports allows assigning a prefix in the context of the respective schema import.

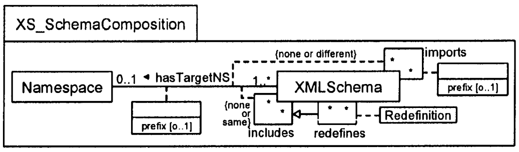


Fig. 5. Meta Schema of XML Schema Composition and Namespace

XS_Types is responsible for providing all necessary information concerning different kinds of types supported by XML Schema, comprising the differentiation in built-in types, i.e., *predefined types* versus *user defined types* as well as in *simple types* versus *complex types*, expressed by inheritance relationships (cf. Fig. 6). Predefined types comprise, for instance, string, integer, date, but also some special ones like anyURI to represent URIs and QName (qualified name) to specify a name that may have a namespace prefix. Simple types specify domains for atomic values, whereas complex types specify domains that hold nested elements, for instance.

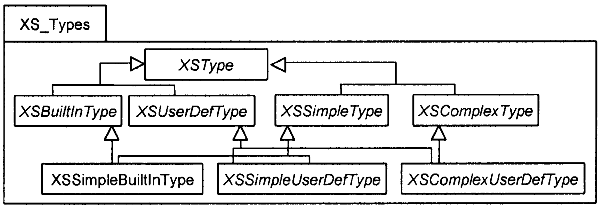


Fig. 6. Meta Schema of XML Schema Types

Resultant combinations supported by XML Schema, which are depicted in Fig. 6 by means of appropriate specializations, are simple built-in types (`XSSimpleBuiltIn Type`) as well as simple and complex user defined types (`XSSimpleUserDefType` and `XSComplexUserDefType`, respectively). Concrete occurrences of the latter two are described in corresponding sub packages.

`XS_SimpleUDTypes` allows specifying alternatives of simple user defined types, expressed by specializations of `XSSimpleUserDefType` in Fig. 7. All these alternatives are connected to `XSimpleType` by respective relationships, denoting different roles the type may play. The alternatives are restrictions (`XSRestriction`) of simple types (role `base`), unions (`XSUnion`) of simple types (role `memberType`), and finally, lists (`XSList`) of simple types (role `itemType`). By means of restrictions it is possible to specify enumerations, patterns, and lower and upper limits, for instance. This is in contrast to DTDs, where just enumerations are supported. Further, enumerations in DTDs are restricted to be of type string and may be applied to XML attributes, only. This limitation is reflected in the meta schema part for DTDs 9.

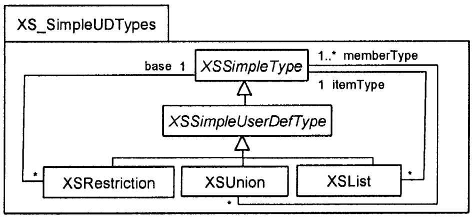


Fig. 7. Meta Schema of XML Schema Simple User-Defined Types

`XS_ComplexUDTypes` allows storing different kinds of complex user defined types, expressed by specializations of `XSComplexUserDefType` (cf. Fig. 8). First, there are composite types with element content (`XSCompositeType ElemContent`) and with mixed content (`XSCompositeTypeMixedContent`), both containing nested elements, whereby the latter in addition contains atomic values in between the nested elements. Second, there are special kinds of types also called complex types, namely types with an atomic value together with attributes (`XSAtomicTypeWithAttr`) and empty types possessing neither element content nor an atomic value (`XSEmptyType`).

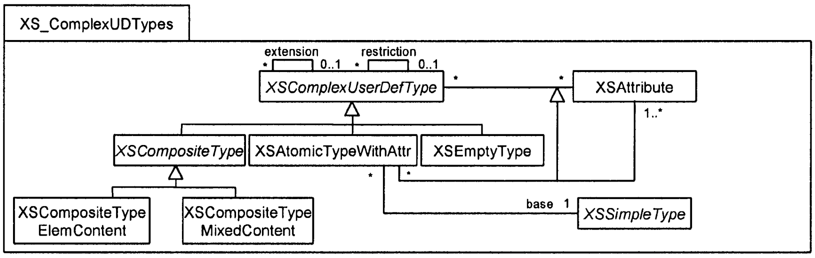


Fig. 8. Meta Schema of XML Schema Complex User-Defined Types

The former holds a relationship to `XSAttribute`, representing a specialization of the general relationship between `XSComplexUser DefType` and `XSAttribute`, in that the actual type must at least obtain one Attribute. Finally, complex user defined types may be extended and restricted, respectively, being depicted in Fig. 8 by recursive relationships of class `XSComplexUserDefType`.

5 Related Work

In 9, we provide an in-depth comparison of X-Ray to thirteen related approaches, among them research prototypes like LegoDB 2, SilkRoute 4, and XTABLES 5 as well as XML support of commercial RDBS, namely DB2 1, Oracle 6, and SQLServer 13. In the following, the most distinguishing characteristics of X-Ray with respect to the compared approaches will be summarized. Most of the approaches hard-code the *mapping knowledge* and about one-third reifies the mapping knowledge within files. Only one approach reifies the mapping knowledge within a database as X-Ray does, thus ensuring mapping transparency and easing maintenance of mapping knowledge. Only a few of the approaches support *multiple schemata* at the DB-side, whereas half of them support multiple schemata at the XML-side. Only one of these approaches supports multiple schemata at both sides, like X-Ray does by allowing multiple relationships between the reified concepts of the schemata to be defined the meta schema. About one-third serves for both, *publishing and storing* of XML documents and thus, provides a unified approach as X-Ray does. Most approaches provide *access* via the XML schema side, one approach allows to access the mapping knowledge as X-Ray does and thus to reason about the mapping knowledge by querying the DB storing the mapping knowledge.

6 Outlook

Future work goes into three different directions. First, XML Schema concepts currently not supported by the meta schema and the prototype, respectively, have to be investigated and incorporated appropriately. Concepts actually not supported by the prototype are complex types with mixed content, simple user-defined types, documentation, notations, the *any* concept, as well as element and attribute groups. Whereas element and attribute groups represent valuable concepts to facilitate reuse and thus, should be incorporated, the any concept enables XML documents or parts thereof to contain arbitrary data, not necessarily conforming to a particular part of an XML schema. Since this is in contrast to the philosophy followed by X-Ray, this concept could be supported by mapping such elements or attributes to single attributes of an RDBS relation, only, instead of decomposing them and mapping them to different attributes.

Second, it has to be elaborated to which extent existing algorithms for the semi-automatic detection of heterogeneities and the generation of subsequent mapping knowledge (cf., e.g., 12) could be employed in X-Ray, to at least partly relieve the user from the burden of defining the mapping knowledge manually. It has to be emphasized, however, that the automatic generation of mapping knowledge, i.e., without user interaction, is problematic in case of schemata developed independently of each other. Such an automatic generation would be especially feasible for the simple case where one schema should be derived from another, already existing schema, which is, however, not the focus of X-Ray.

Third, the approach of X-Ray is currently applied in the realization of ubiquitous web-enabled EIS, i.e., EIS relying on the anytime/anywhere/anymedia paradigm, being context-aware with respect to time, location, device, and user preferences, for instance. The goal is to employ X-Ray to mediate between existing XML-based context and content stored in relational databases 11. With X-Ray, maintainability and changeability of context data could be enhanced, since the mapping knowledge is not hard-coded but rather reified within a meta schema. The meta schema would allow to automatically compose context data out of an RDBS when requested and decompose them when they have to be stored.

References

1. S.E. Benham, IBM XML-Enabled Data Management Product Architecture and Technology, XML Data Management, Native XML and XML-Enable Database Systems, A. Chaudhri, et al. (eds.), Addison Wesley, 2003.
2. P. Bohannon, J. Freire, J. Haritsa, M. Ramanath, R. Prasan, and J. Simeon, Bridging the XML-Relational Divide with LegoDB: A Demonstration, In the Proceedings of ICDE, 2003.
3. A. Deutsch, et al., MARS: A System for Publishing XML from Mixed and Redundant Storage, Proc. of the Int. Conf. On Very Large Databases (VLDB), Germany, 2003.
4. M.F. Fernandez, et al., SilkRoute: A Framework for Publishing Relational Data in XML, *ACM Transactions on Database Technology* **27**(4), (2002).
5. J. Funderburk, G. Kiernan, J. Shanmugasundaram, E. Shekita, and C. Wei, XTABLES: Bridging Relational Technology and XML, *IBM Systems Journal* **41**(4), (2002).
6. U. Hohenstein, Supporting XML in Oracle9i. XML Data Management, Native XML and XML-Enable Database Systems, A. Chaudhri, et al. (eds.), Addison Wesley, 2003.

7. G. Kappel, E. Kapsammer, S. Rausch-Schott, W. Retschitzegger, X-Ray - Towards Integrating XML and Relational Database Systems, In the Proceedings of the 19th International Conference on Conceptual Modeling (ER), LNCS 1920, Springer, USA, October, 2000.

8. G. Kappel, E. Kapsammer, and W. Retschitzegger, Architectural Issues for Integrating XML and Relational Database Systems – The X-Ray Approach, In the Proceedings of the Workshop on XML Technologies and Software Engineering, Toronto, May 2001.

9. G. Kappel, E. Kapsammer, and W. Retschitzegger, Integrating XML and Relational Database Systems, *World Wide Web Journal* 7(4), 343-384 (2004).

10. G. Kappel, E. Kapsammer, and W. Retschitzegger, XML and Relational Database Systems – A Comparison of Concepts, In the Proceedings of the 2nd International Conference on Internet Computing (IC), CSREA Press, Las Vegas, USA, June 2001.

11. E. Kapsammer, et al., Bridging Relational DB to Context-Aware Services, In the Proceedings of the Workshop on Ubiquitous Mobile Information Systems (UMICS), Portugal, June 2005.

12. E. Rahm and P. A. Bernstein, A Survey of Approaches to Automatic Schema Matching, *VLDB Journal*, **10**(4), 334-350 (2001).

13. M. Rys, XML Support in Microsoft SQL Server 2000, Native XML and XML-Enable Database Systems, A. Chaudhri, et al. (eds.), Addison Wesley, 2003.

14. M. Schrefl, M. Bernauer, E. Kapsammer, B. Pröll, W. Retschitzegger, and T. Thalhammer, Self-Maintaining Web Pages, International Journal of Information Systems, **28**(8), (2003).

15. W3C, Namespaces, January 1999; http://www.w3.org/TR/1999/REC-xml-names-19990114.

16. W3C, XML 1.0 (2nd ed.), October 2000; http://www.w3.org/TR/2000/REC-xml-20001006.

17. W3C, XML Schema, May 2001; http://www.w3.org/XML/Schema.

18. W3C, XQuery 1.0, May 2003; http://www.w3.org/TR/xquery.

19. Y.-H. Yao, A. J. C. Trappey, and P.-S. Ho, XML-based ISO9000 Electronic Document Management System, *Robotics and Computer Manufacturing* **19(4)**, (2004).

20. D.C. Yen, S.-M. Huang, and C.-Y. Ku, The Impact and Implementation of XML on Business-to-Business Commerce, *Computer Standards & Interfaces*, **24**(4), (2002).