ShaRef: XML-Centric Software Design

Erik Wilde, Sai Anand, Petra Zimmermann
ETH Zürich (Swiss Federal Institute of Technology)

TIK Report 213 (February 2005)

Available at http://dret.net/netdret/publications#wil05k

Abstract

In this paper, we describe a real-life application which has been designed around an XML data model and various XML technologies. We describe the rationale behind this design, and the benefits from the information system design point of view. We believe that XML-centric design has a lot of benefits, and that future developments in the area of XML technologies will better support this design style and help to make it even more advantageous. XML-centric design allows to leverage an ever-increasing number of XML-based technologies. We describe some of the XML technologies that helped us implementing some of the core parts of the software, and point out some others that we do not yet use, but are actively investigating for possible future developments.

1 Introduction

The *Extensible Markup Language (XML)* [4] is generally accepted as an exchange format. Increasingly, XML and its companion technologies are also penetrating applications, and XML-oriented ways of designing interfaces and constructing software are becoming more important, as demonstrated by *SQL/XML* [11], which integrates XML as a first-class data type into the SQL model.

While XML is certainly successful in the data exchange and data structuring area, it is not as successful as a general foundation for modelling structured data, as described in more detail in Section 2. The main reason for this is the absence of a generally accepted conceptual model for XML.

Traditionally, researchers and scholars have maintained personal bibliographies using various tools, for instance BibTeXand EndNote. In the *Shared References (ShaRef)* project [25], the aim is to develop a tool that helps maintaining bibliographic information in individual and group settings, so that bibliographies can be shared between members of the same group or — in the case of inter-disciplinary fields or larger projects — between several groups. In the ShaRef Project, the data model (in Section 3) is based on XML Schema. This enables us to easily define and implement import and export filters (Section 5). ShaRef has been designed to use as much XML technologies as possible, and we have found that this not only makes software development easier and faster, but it also helps to create reusable software components. These components can be reused in many different places, and in case of XSLT programs even as script-like programs that can be reused in almost any execution environment, totally independent from ShaRef or a Java run-time environment.

In this paper, we do not go into the details of any of the involved technologies. Instead, we give a tour of all places where we used XML technologies, in the data modelling itself (Section 3), for managing and storing data (Section 4), for importing and exporting data (Section 5), and for designing and implementing interfaces (Section 6).

2 XML-Based Modelling

As pointed out in the introduction, XML has started as an exchange format and is increasingly penetrating applications. The reason for this is simple: If applications are faced with an ever increasing number of components with which they must or at least can interact using XML, it makes sense to align an application's data model with XML, so that the XML-based interactions can be implemented as easily as possible.

Traditionally, XML-based modelling was done using the *Document Type Definition (DTD)* mechanism, which is an integral part of XML. With the increasing popularity of XML, DTDs were quickly considered as being too limited, and after a period of consolidation, *XML Schema* [2, 24] appeared as the new schema language for XML.

XML Schema has a number of higher-level modelling mechanisms that were absent in DTDs, most importantly the type derivation mechanism, and reusable groups. The whole type hierarchy in XML Schema is something that is only important for modelling, it does not necessarily influence the surface of the schema (i.e., the types that will be instantiated in documents). On the other hand, XML Schema still requires a schema author to address fairly low-level XML syntax issues, such as the namespace qualification of locally defined names. Thus, XML Schema combines two different modelling issues:

- *Conceptual Modelling:* In conceptual modelling, the goal is to define useful abstractions and relations between abstractions. XML Schema's complex type derivation is very useful for doing this, but on the other hand for relations it must be decided whether these should be represented through containment or references. Therefore, even though defining a type hierarchy provides a useful level of abstraction, it also requires decisions about implementation issues.

- *Logical Modelling:* Since XML Schema describes the syntax of documents, it needs to define XML-specific things such as whether a particular piece of information is represented as an attribute or an element. Other decisions that are very syntax-specific are the decision how to implement type substitution (`xsi:type` vs. substitution groups), the definition of simple type facets, and the question how to qualify local names.

  The most important logical modelling question that must be addressed with XML Schema is the question how to model relationships. While conceptual modelling typically only has generic relationships, qualified by the allowed cardinalities, XML can represent relationships through containment (which can only represent 1:n relations) or (in most cases attribute-based) referential information. While both ways of representing relationships are structurally isomorphic, they have very different implications for the structure of XML documents and thus are important design decisions from the XML point of view.

Since XML Schema authors often want to have a more abstract way of looking at a schema (and because XML Schema's XML syntax is not very readable), XML tools such as

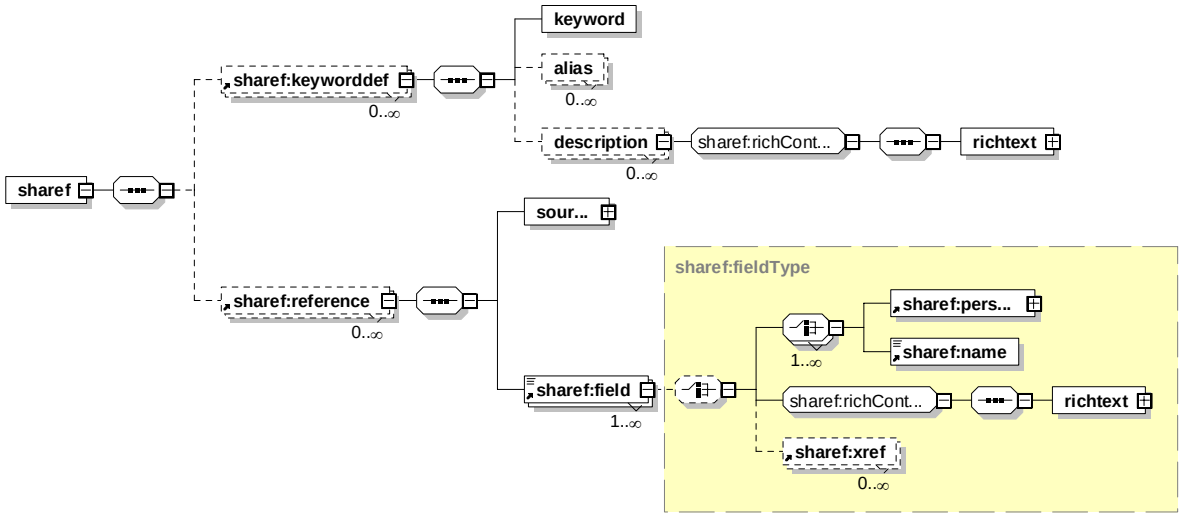

Figure 1: ShaRef Schema

XMLSpy™ or *Turbo XML*™ have implemented their own graphical representations of XML Schemas[1]. However, since these tools are not really modelling tools, but rather XML Schema editors, they also have to make every possible detail of XML Schema available through the graphical representations and the associated interface, which severely limits its usefulness as a conceptual modelling tool.

Since there is no standardized or generally accepted conceptual modelling language for XML, research projects such as *ER extended for XML (EReX)* [17], *X-Entity* [15], and *Extensible Entity Relationship Modeling (XER)* [23] have defined extensions to the well-known ER model [6]. However, so far none of these languages have reached a critical mass where they are widely accepted and used.

Coming from the UML modelling world, the *XML Metadata Interchange (XMI)* [21] language is a method for expressing UML models in XML, but it is not geared towards modelling for XML Schema. XMI's focus is representing UML models, and it is possible to map XMI to XML Schema. However, XMI-derived XML Schemas tend to suffer from poor readability (from the XML Schema point of view), and so far there is no well-established way how to generate a (logical) XML Schema from a (conceptual) UML/XMI model. As a consequence, the ShaRef data model is defined directly in XML Schema.

3 ShaRef Data Model

The ShaRef data model shown in Figure 1 as its cornerstones has keywords and references. Keywords can be accompanied by any number of aliases and any number of descriptions and are used for classifying references. ShaRef does not have any concept of structuring keywords (into hierarchies, taxonomies, ontologies, or any other form of structure). The more important part of ShaRef are the references, which contain the actual reference information (they can be thought of as the equivalent of BibTeXentries or EndNote references).

[1]The XML Schema representations included in this paper have been produced with XMLSpy.

Each reference comes from a particular source and has a number of fields. The data model uses a generic `field` element, which is head of a substitution group containing all concrete field elements (such as elements for title or author information). Since concrete field elements can have different content types, the generic `field` element uses a generic `fieldType`, which for the concrete field elements is derived by restriction into four different field types:

- `nameFieldType`: Fields containing name information, such as information about authors and editors.
- `richFieldType`: Fields containing rich textual information, such as annotations.
- `xrefFieldType`: Fields containing cross-references, such as references from a paper to the proceedings.
- `simpleFieldType`: Fields with simple textual content, such as dates or pages.

While this is modelled in XML Schema using XML Schema's type restriction and type substitution features, the general model on the reference level is mainly relational (i.e., more strongly structured that semi-structured). However, the `richFieldType` shown in Figure 2 is more interesting, because it is a more document-oriented structure, being composed out of paragraphs which may use a small number of text formatting elements, and two kinds of structural information, one being hypertext links to any URI, and the other being internal links to other references or keywords. Even though this structure is irregular and rather flexible, it is important to notice that it is of very limited depth, because we do not allow nesting of the constructs. This allows us to map rich text to simple text-formatted representations, which is described in more detail in Section 6.1.

The data structures regarding user and group management and access authorization are not described here, but they are also part of the data model and important for implementing the multi-user facilities of the system.

An interesting feature of the ShaRef Data Model is the flexibility of including many different types of user-defined fields or information that does not fit into the pre-defined fields of the model. This is incorporated through the use of the `field` element[2] to hold generic information. The user may qualify newly introduced fields using namespace-qualified attribute values for the `field` element. Among the advantages of this approach are an almost lossless information exchange during the import/export process, and custom processing of the new fields through (user-extensible) XSLT scripts.

It is also pertinent to compare our model with that of the *Metadata Object Description Schema (MODS)* schema[3], an XML Schema developed by the Library of Congress' Network Development and MARC Standards Office. MODS is an XML Schema for mapping a subset of MARC-standard numeric fields to the schema defined tags. While MODS considerably dilutes the complexity of the MARC standard, it is still too generic and complex for maintaining individual bibliographies. Our model is able to reduce this genericity by taking a near optimal intersection of the two import formats (BibTeXand EndNote) we support. This makes our model much simpler to understand and use than MODS. On the other hand, the

[2] Technically, the `field` element is the substitution group head for all specific field elements. This can be seen in Figure 1, which only shows the substitution group head `field` as permitted content for the `reference` element.

[3] http://www.loc.gov/standards/mods/

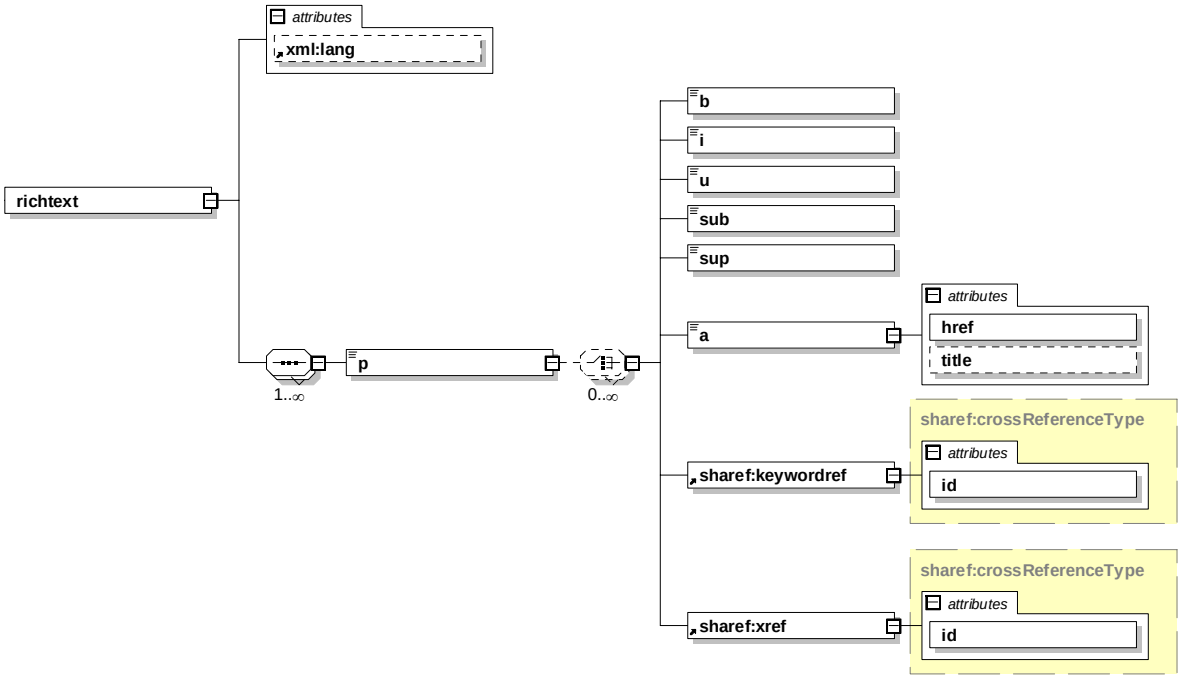

Figure 2: Schema for Sharef Rich Text

accommodation of user defined type (as mentioned in the last paragraph) yields a lot of flexibility, and enables users to losslessly round-trip MODS bibliographies through ShaRef.

4 Data Storage

ShaRef supports online and offline modes. In online mode, the client acts as a rich client, communicating with a server that holds all the data. In offline mode, the client uses a local database, thus acting as a standalone application. In both cases, the data needs to be stored and queried efficiently, so some kind of database support is required. Furthermore, the client implements a workspace metaphor, which serves as a "shopping cart" for references, holding references (either from online or offline storage) which can be used to perform specific operations, such as exporting, publishing, or searching. Users can have multiple workspaces, for example using different online storages, or using an online as well as their private offline storage.

Looking at these scenarios and the implications for data storage design, the following issues are important:

- *Client-side Cache:* For the workspace model to work efficiently, each workspace is supported by a database, mirroring the database structure of the persistent storage. The workspaces, however, use in-memory tables, improving performance and thus making it possible to reuse existing DBMS code.

- *Client-side Storage (Offline Mode):* In offline mode, the database must be available locally. As the client cannot depend on a local DBMS being available, the DBMS must

be part of the client. This places serious restrictions on the DBMS, since it must be fairly small, cross-platform, and freely distributable.

- *Server-side Storage (Online Mode):* In online mode, the client serves as a rich client to the server, which is an application server incorporating the DBMS. Since ShaRef is a multi-user system, the DBMS on the server side must support multi-user access, and typical multi-user features such as locking and transactions.

The most important question is whether to use a relational DBMS or an XML DBMS. While the data model described in Section 3 is defined in XML and thus would lend itself easily to being supported by an XML DBMS, it is also apparent that the model is more strongly structured than typical document-oriented XML applications, and thus also can be supported by a relational DBMS.

Because of the requirements for in-memory storage, a small footprint, and support for both offline and online access, the relational DBMS approach was chosen. Recalling the discussion from Section 2, this is the decision to use an RDBMS on the physical modelling layer, even though the logical modelling layer uses an XML Schema. Again, various tools provide out-of-the-box solutions for deriving relational schemas from XML Schemas, but mapping XML structures to relational structures is better done by hand or a sophisticated methodology, as explained by Pankowski [22]. Since the ShaRef data model is a tree, mapping it to a relational structures on the physical modelling layer is rather easy and required no potentially dangerous table designs.

We use *HSQLDB*[4], a small pure Java relational DBMS. HSQLDB supports a subset of SQL. On the server side, the database software is less central, because it can run separately and is simply accessed through a JDBC driver. HSQLDB is not sufficient for the server side, because it does not provide support for multi-user scenarios requiring locking and transactions.

The JDBC layer allows us to hide he fact that we are using different databases on the client and on the server side. As long as references are only transferred from the online or offline database to the workspace, they do not need to be interpreted as XML, they are simply copied from one database to another. However, if references are selected to perform operations on them, such as exporting (Section 5) or editing (Section 6), then they are selected from the database and transformed into XML objects.

5 Import/Export

One of the advantages of the XML-based data model is the availability of XML technologies for importing and exporting data. While importing data is limited to XML-based data, exporting can be done to every text-based format. As a natural choice for transforming XML, we use *XSL Transformations (XSLT)* [12], in particular the upcoming version 2.0 of the language, which has been greatly improved in a number of areas. Figure 3 shows the general structure of the import/export process, and also shows that a special piece of software is used for parsing BibTeXfiles, which is the *javabib*[5] parser. This is necessary because BibTeXuses a text-based and rather idiosyncratic syntax.

BibTeXfiles are parsed and the BibTeXparse tree is mapped into an XML DOM tree, without any structural modifications, it is a 1:1 XML equivalent of the original BibTeXstructure.

[4] http://hsqldb.sourceforge.net/
[5] http://www-plan.cs.colorado.edu/henkel/stuff/javabib/

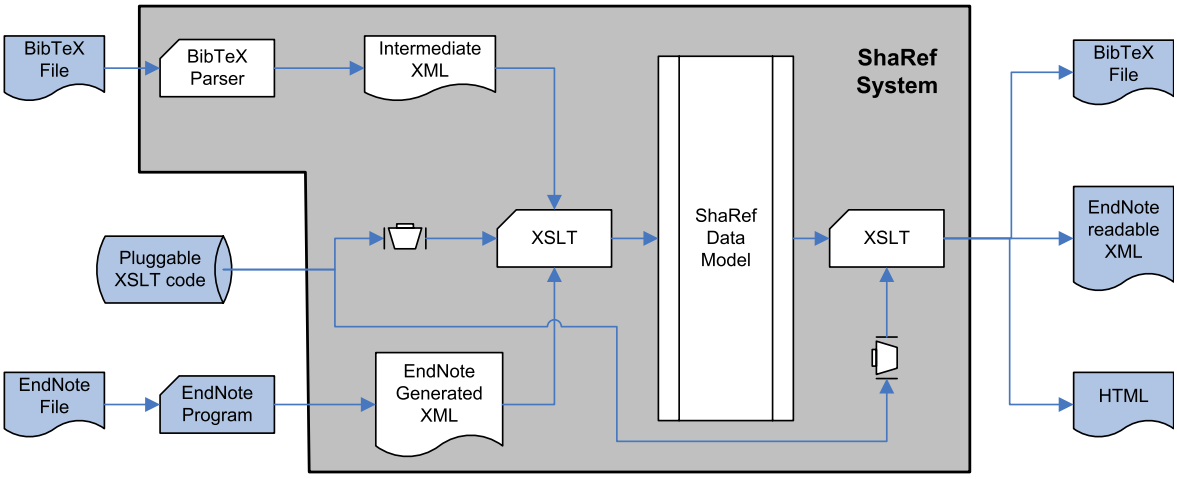

Figure 3: Import/Export in ShaRef

This BibTeX-oriented XML is then transformed using an XSLT program. This way, the BibTeXparser could be reused without any modifications, and the structural transformations are done using XSLT. The advantage of this approach is that all mapping information can be kept and maintained in a single place, an XML mapping table that keeps all the information about mapping all supported import and export formats. This mapping table serves as a configuration information for the import/export transformations, as well as for generating the documentation.

The import of user maintained bibliographies in EndNote is simpler than for BibTeX. The EndNote program provides an option to export the bibliography into an XML format. The EndNote generated XML can then be processed by an XSLT script to convert it into an XML that conforms to the ShaRef data model. Figure 3 shows this processing, including an optional step for additional user-defined processing.

Since XSLT is an interpreted language, it is easy to provide users with pluggable import/export filters. Using XSLT's `include`, these can be easily plugged into import or export XSLT code, and can be used to perform specific import or export tasks. While this requires XSLT programming, most import/export tasks can be handled with very few lines of code (Keith [13] describes in detail how to process bibliographic metadata using XSLT). The cost of implementing the plug-ins is often offset by the benefit of being able to use custom-made import and export filters.

As a special case of "export" (and therefore not shown in Figure 3), we also use XSLT to construct *OpenURL* [1] references, which basically are URIs containing encoded bibliographic metadata. By configuring their own library's OpenURL server in the ShaRef client, users are provided with OpenURLs pointing to their local library, which in most cases will lead them directly to the library's catalog data and maybe even fulltext resources for their bibliographic references.

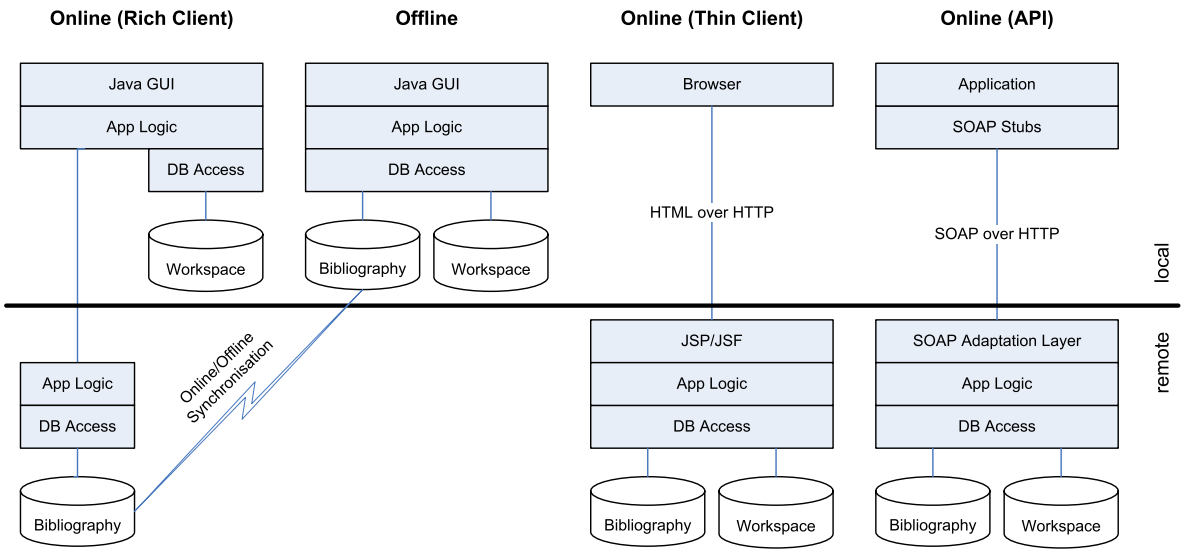

Figure 4: Local/Remote Distribution for ShaRef Client and Service Scenarios

6 Interface Issues

The data model and the import/export features are useful in their own right, because they define and implement a unified view of the diverse world of (bibliographic) references[6]. Most users, however, need tools to work with references, and this is what the actual ShaRef software does. Figure 4 shows how the ShaRef system is designed to be used in a variety of scenarios, ranging from purely local, using a Java-based application, to a Web Service, which is accessed from remote applications.

ShaRef supports three different interfaces, two of them being user interfaces (one is Java-based for online and offline access, the other one is Web-based), which are described in Section 6.1, and the other being an application interface (based on Web Service technologies), described in Section 6.2. All of them provide different views of the ShaRef data model, because of the specific interface technologies, but the overall goal has been to stay as close to the native XML model as possible, with the goal of avoiding interface inconsistencies.

6.1 User Interface

The initial implementation of ShaRef uses a Java-based rich client, which — as discussed in Section 4 — operates either in offline or online mode. In both cases the data is retrieved from a data source (the online or offline database), placed into the workspace database, and then ready for manipulation by the user. The main view of the Java-based user interface is the table view shown in Figure 5. This view lets users select from multiple data sources, and displays the workspace for all these data sources.

The table view is a simplified view of the actual data, because it only displays textual data. The references' deeper structures, as shown in Figure 1, are not visible in this view. Using XPath's concept of the *string value* of nodes, the table view can be described as listing

[6] `http://bibliophile.sourceforge.net/` provides a comprehensive overview of tools for managing bibliographic references.

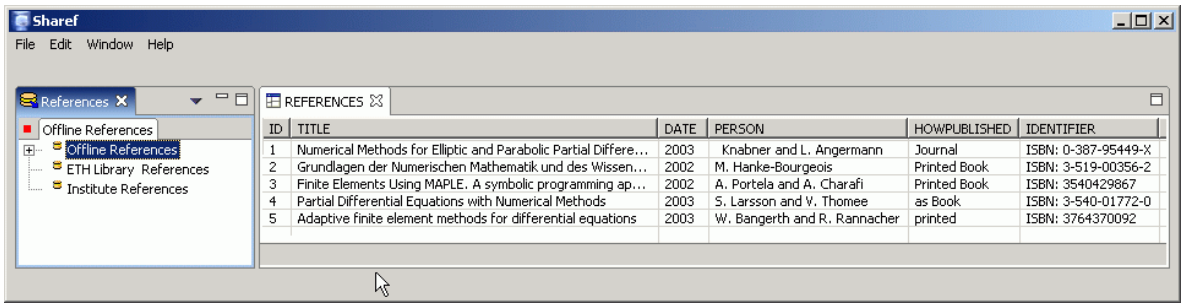

Figure 5: Table View in the ShaRef Rich Client

the string values of the individual field elements of each reference. When selecting table rows for performing operations, however, the full structure is used, so structured fields such as names or rich text are processed according to their internal structure.

While the Java-based rich client can provide convenient ways of dealing with structured fields, this is more problematic for the Web-based version of ShaRef, which uses an interface based on *HTML Forms*. To enable the use of structured content through the Web-based interface, a simplified version of the WikiWikiWeb [14] text formatting rules is used, which provide the full functionality of ShaRef's structured fields. Using the tokenization functions from XPath/XSLT 2.0 [12,16], mappings of the XML structures and the text-based structures can be implemented easily.

While the current Web-based version is based on simple HTML forms, the XML data model of ShaRef would make it very easy to advance to the next generation of Web forms, the XML-based *XForms* [8]. XForms promote a separation of the underlying data model and an interface supporting this data model, and thus is an ideal candidate for providing a better Web-based interface to ShaRef than the HTML form approach. Unfortunately, support for XForms is not widely implemented, and because of this we decided to support the simpler HTML forms instead of the more powerful XForms.

6.2 Application Programming Interface

ShaRef is primarily designed as a program that can be used in online and offline modes. However, ShaRef also is a service that can be accessed through an API (as shown in Figure 4). The motivation for this is that institutional users of ShaRef don't want to use the GUI, but want to be able to integrate ShaRef in their own applications. Being interested in a loose coupling between the ShaRef service and prospective API users, we decided to define the API as a *Web Service*. Thus, the ShaRef API is defined using the *Web Service Description Language (WSDL)* [7], and messages are sent using the *Simple Object Access Protocol (SOAP)* [9].

One big advantage of the XML-based data model described in Section 3 is that it can be directly reused in the API definition. While WSDL theoretically accepts any XML schema language, in practice all WSDL definitions use XML Schema type definitions. We can thus directly use the ShaRef XML Schema, without any need to create mappings to other data models that could result in mapping problems and make the overall architecture harder to understand.

By using ShaRef's native data model for the API, we enable a perfect alignment of the

different interface worlds, regardless whether ShaRef is accessed through a GUI, through its API, or whether data is exported from ShaRef, in which case the data model will be directly exposed as an XML document.

7 Discussion

In this paper, we describe an XML-based data model and a number of places where we benefit from the opportunity of an XML-centric approach. While XML as a format for data exchange has had enormous success in the last years, it still is the exception to see XML as a core component of an application architecture. In the following discussion, we mention some related work (Section 7.1), point out possible future work (Section 7.2), and highlight the contributions of our paper (Section 7.3).

7.1 Related Work

XML-centric design is still in its early stages. XML-centric methods are developed in different areas, such as the *Abstract User Interface Markup Language (AUIML)* [18], which enables developers to use a single interface description as foundation for a Java-based as well as a Web-based GUI. In the modelling world, there still is no widely accepted conceptual modelling language for XML, so that approaches such as EReX [17], X-Entity [15], XER [23], or the XMI adaptation to XML Schema as suggested by Carlson [5], are possible ways to handle data modeling in XML-centric applications. We have decided to use XML Schema directly, because it provides minimal modelling support, and because ShaRef's data model is not very complex.

In the context of the *Model-Driven Architecture (MDA)* [19], two modelling layers are defined, the *Platform-Independent Model (PIM)*, and the *Platform-Specific Model (PSM)*. Even though MDA does not prescribe any particular modelling language, the *Unified Modeling Language (UML)* [20] is the most widely used language for both the PIM and the PSM layer. Using Carlson's method, it is possible to use an UML profile that adds XML Schema specific to the model for going from the PIM to the PSM. While this approach limits the flexibility from the XML Schema point of view, it guarantees the consistency of the PIM and the PSM, by deriving the latter from the former.

7.2 Future Work

While the first beta release of ShaRef is still under construction, we already have some plans how to extend ShaRef with new services and capabilities. The following list contains only items that are relevant from the XML point of view.

- *User-defined Queries:* So far, queries can only be formulated using a simple form-based query interface, that allows to search for references by simple search criteria. Because the underlying data model is using XML, it would be possible to allow users to formulate queries using the upcoming *XML Query Language (XQuery)* [3]. However, the current implementation of ShaRef uses a relational DBMS for storing references, and in order to support XQueries, it would be necessary to switch to an XML DBMS supporting XQuery[7].

[7] Technically, SQL/XML also includes XQuery, and therefore XQuery should be supported by RDBMS

- *Catalog Access:* Apart from entering references by importing them or manually entering them, it would also be useful to provide catalog access from with ShaRef (this feature is also provided by EndNote). Library catalog access usually is based on the *Z39.50* [10], which uses ASN.1 for encoding data. More modern adaptations of this protocol use newer transport mechanisms, in particular the *Search and Retrieve Web Service (SRW)*, based on Web Service technologies, and the *Search and Retrieve URL Service (SRU)*, which uses URL-encoded data. It is still unclear how fast SRW and SRU will be adapted by libraries, but if they become available, they will over a much more convenient way to access library catalogs than the ASN.1-based version of Z39.50.

7.3 Contributions

In this paper, we show the advantages of an XML-centric design for a number of important software design issues. For storing data, we choose a relational approach on the physical level, but the interfaces to the application logic use the XML-based logical model. We describe the benefits for importing and exporting data, which is very important for the ShaRef project because the software must be open to extensions and should not lock users into its data model. For the interface, we both show advantages for the user interface design, where we can easily map our data model to a Web-based interface, and for the application programming interface, which directly uses the types defined in the data model. To summarize, even though there currently is no unifying overall framework or model for XML-centric application design and implementation, the benefits are considerable and interesting for applications which have many interfaces to XML-based technologies.

8 Conclusions

Many XML technologies are still in the developing stage, and it will take some time until there is a consistent, time-tested, universally accepted framework for XML-centric application design and development. Until then, application designers and developers are mostly on their own when it comes to choosing the most appropriate technologies for designing and implementing XML-oriented information systems. In this paper, an example application is used to demonstrate the benefits of XML-centric application design. While we do not claim that the approach described in this paper is advantageous for all types of applications, the special constraints of the ShaRef project, most notably an open and extensible data model, and an application architecture supporting online and offline GUIs as well as an API, made this design a reasonable and beneficial choice.

References

[1] American National Standards Institute. The OpenURL Framework for Context-Sensitive Services. Part 1: ContextObject and Transport Mechanisms. ANSI/NISO Z39.88-2003 (Draft), March 2003.

products implementing SQL/XML. In reality, however, XQuery is still under construction, and it is unclear when full XQuery support will be provided for RDBMS products.

[2] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-2-20041028, October 2004.

[3] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Working Draft WD-xquery-20050211, February 2005.

[4] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). World Wide Web Consortium, Recommendation REC-xml-20040204, February 2004.

[5] David Carlson. Integrating XML and non-XML Data via UML. In *Proceedings of XML 2001*, Orlando, Florida, December 2001.

[6] Peter Pin-Shan Chen. The Entity-Relationship Model — Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[7] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language. World Wide Web Consortium, Working Draft WD-wsdl12-20030611, June 2003.

[8] Micah Dubinko, Leigh L. Klotz, Roland Merrick, and T. V. Raman. XForms 1.0. World Wide Web Consortium, Recommendation REC-xforms-20031014, October 2003.

[9] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. World Wide Web Consortium, Recommendation REC-soap12-part1-20030624, June 2003.

[10] International Organization for Standardization. Information and Documentation — Information retrieval (Z39.50) — Application service definition and protocol specification. ISO 23950, 1998.

[11] International Organization for Standardization. Information Technology — Database languages — SQL — Part 14: XML-Related Specifications (SQL/XML). ISO/IEC 9075-14, December 2003.

[12] Michael Kay. XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, Working Draft WD-xslt20-20050211, February 2005.

[13] Corey Keith. Using XSLT to Manipulate MARC Metadata. *Library Hi Tech*, 22(2):122–130, 2004.

[14] Bo Leuf and Ward Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison Wesley, Reading, Massachusetts, April 2001.

[15] Bernadette Farias Lósio, Ana Carolina Salgado, and Luciano do Rêgo Galvão. Conceptual Modeling of XML Schemas. In Roger Chiang, Alberto

H. F. Laender, and Ee-Peng Lim, editors, *Proceedings of the 5th ACM International Workshop on Web Information and Data Management*, New Orleans, Louisiana, November 2003.

[16] Ashok Malhotra, Jim Melton, and Norman Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators. World Wide Web Consortium, Working Draft WD-xpath-functions-20050211, February 2005.

[17] Murali Mani. *Data Modeling using XML Schemas*. PhD thesis, University of California, Los Angeles, 2003.

[18] Roland Merrick, Brian Wood, and William Krebs. Abstract User Interface Markup Language. In Kris Luyten, Marc Abrams, Jean Vanderdonckt, and Quentin Limbourg, editors, *Proceedings of the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages*, Gallipoli, Italy, May 2004.

[19] Object Management Group, Framingham, Massachusetts. *MDA Guide Version 1.0.1*, June 2003.

[20] Object Management Group, Framingham, Massachusetts. *OMG Unified Modeling Language Specification Version 1.5*, March 2003.

[21] Object Management Group, Framingham, Massachusetts. *OMG XML Metadata Interchange (XMI) Specification Version 2.0*, May 2003.

[22] Tadeusz Pankowski. XML-SQL: An XML Query Language Based on SQL and Path Tables. In Akmal B. Chaudhri, Rainer Unland, Chabane Djeraba, and Wolfgang Lindner, editors, *International Conference on Extending Database Technology — EDBT 2002 Workshop on XML-Based Data Management (XMLDM)*, volume 2490 of *Lecture Notes in Computer Science*, pages 184–209, Prague, Czech Republic, March 2002. Springer-Verlag.

[23] Arijit Sengupta. XER — Extensible Entity Relationship Modeling. In *Proceedings of XML 2003*, Philadelphia, Pennsylvania, December 2003.

[24] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.

[25] Erik Wilde. References as Knowledge Management. *Issues in Science & Technology Librarianship*, (41), Fall 2004.