XML-Centric Application Development

Erik Wilde
ETH Zürich (Swiss Federal Institute of Technology)

February 2006 (TIK Report 242)

Available at http://dret.net/netdret/publications#wil06c

Abstract

XML has become an important standard for exchanging structured data between applications, but XML increasingly penetrates applications and is thus also becoming an important part of application development. The current state of XML specifications and technologies provides support in many aspects of application development, while other aspects are still only poorly supported. We describe as an example the development of an XML-centric application and identify and describe the areas where today's support for application developers could and should be improved. This case study thus can help developers to focus on the problem areas of today's support for XML-centric application development, and may also serve as an agenda for areas where more research and tools are required to improve the development of XML-centric applications.

1 Introduction

XML has become an important foundation for data exchange, and XML support in some form is available in almost all application development environments. XML's success is based on its success as a platform- and language-neutral format for representing structured data, and thus XML's usage and success started on the very outside of applications, where data has to be generated or consumed for cross-application data exchange. Over the past years, however, XML technologies have steadily moved towards the core of applications, which is nicely demonstrated by how XML support has evolved in database systems:

1. *Hand-coding:* When XML appeared, database systems had no support for XML at all, so any XML data that has to be retrieved from or stored in a database had to be manually mapped to relations. For data-oriented XML, this was rather easy, for document-oriented XML, this was hard to practically impossible. (This method is often called "XML-enabling" a database, in this case on the application level.)

2. *Shredding:* When XML became popular, database vendors provided proprietary support for handling XML. This often included configurable ways of how XML should be generated from queries, and how XML data should be distributed among different tables of a database. Again, this solution worked well for data-oriented XML, but not so well for document-oriented XML. (In this case, the database system was "XML-enabled" through middleware in the database system itself.)

3. *SQL/XML:* Recognizing the growing need for XML support, the SQL standard itself was extended to support XML, the result was called SQL/XML [9]. The most important development was that XML was introduced as a basic datatype of an SQL database, so users can store XML in columns. SQL/XML also introduced support for retrieving the result of an SQL query as XML. Since XML is now supported as a datatype, document-oriented XML can be easily inserted into the database.

4. *XQuery:* While SQL/XML stays within the relational model, the upcoming *XML Query Language (XQuery)* [2] assumes an underlying database system which simply stores XML documents. These so-called *native XML databases* can be regarded as a specialized file system for XML documents with performance optimizations for storing, querying, and retrieving XML.

5. *XML Database Systems:* While XQuery is a first step towards XML database systems, today there are still many open research issues, for example a data definition and an update language. XML Schema [17] has been designed as a schema language for documents, not for databases. As a database schema language, it lacks essential features such as a powerful and flexible model for inter-document integrity constraints [5], and support for schema evolution.

While it is an open question whether native XML databases will acquire a large share of the database market, it is undoubtedly true that an increasing amount of XML has to be stored in databases, and that XML support thus will become an essential functionality of many database systems. The next logical question is: if the database system supports XML as the native data format, what about the application? Looking at today's state of the art of XML support in programming languages, most languages support basic functionality such as parsing, validating and handling document trees, and maybe even transforming XML documents, but currently there is little or no support for operations directly on the underlying data structures represented by the XML. In Section 3, we review what this means for developing XML-centric applications in the currently available software environments, and what pieces we identified as missing which would greatly improve and ease the use of XML in application development.

While the core of an information system, the data model and the operations on it, are essential for the system's functionality, another important determining factor are the available interfaces. With interfaces we refer to anything that provides access to the system's data. In this paper, we present as a case study an information system for the management of bibliographic metadata, and we concentrate on the issues that arose during system design as a consequence of the XML-based data model and the different interfaces we wanted to support.

The first issue was caused by the fact that we wanted to built a system around an XML-based data model. The reason for this decision was the fact XML is a good candidate for the kind of data we wanted to manage (metadata in a mostly "relational" structure with some additions of document-style content for some fields of the bibliographic records), that XML technologies provided good support for manipulating this kind of data, and that we wanted to have a low barrier-to-entry (for example for users creating their own import and export filters). Simpler text-based formats would not have supported the document-style content, while more complex formats (such as RDF-based formats) would have made it harder for users to work with the data.

The second issue (supporting different interfaces) was important because our initial user survey discovered two different user groups: The first group wanted to have a stand-alone tool they could use as their "personal information management tool", which should have additional collaboration support which could be used when required. The second group wanted Web-based access to centrally stored data, so that they did not have to store any data locally. In order to satisfy both user groups, we decided to build a Java-based application which could be used as a standalone application or in collaboration mode, as well as a Web interface to the server which provided (almost) the same functionality as the standalone client.

In this paper, we report on how system development was affected by the two design decisions described above (XML-based data model and two user interfaces), and what is missing in the current landscape of XML specifications and tools to better support this kind of system development. The most important observation is that there currently is no methodology of how to specify and develop a conceptual model of an XML-based system. While the grand vision of software development built around executable conceptual models as laid out by Olivé [14] may be too much to ask for at the moment, at least some support for the conceptual modeling of XML-based application would be helpful.

2 Requirements

The system we use as the starting point for this experience report is an application for the management of bibliographic metadata, it is called *Shared References (ShaRef)*. The focus is on research settings, where researchers are collaboratively working with bibliographic metadata and would like to share and reuse this data. Since requirements tend to be more stable than the information in a conceptual model [16], the project started with an evaluation of the requirements of future users of the system.

The starting point of the project was a survey among researchers at the local university [19], which on the one hand showed that there was demand for such a tool, and on the other hand produced feedback which showed which features were considered essential or at least important by the future users. By combining the initial idea for the system and the user feedback, the following requirements were identified for the system:

- *Open System Design:* The system should provide easy migration paths to and from the system. System lock-in must be avoided, and users must be able to retain the feeling of "their" data. Most users are using BibTeX and EndNote, and these formats must be fully supported for import and export, avoiding the information loss often occurring during import and export.

- *Rich and Web Clients:* The goal of the system is to provide a collaborative environment as well as features which enable researchers to use it as their personal information management tool. Thus, the system should be available as a stand-alone system, as well as through a Web interface which allows access to shared data without the need to install any software locally.

- *Offline Mode for Rich Client:* The rich client (the stand-alone system) should provide access to shared as well as to local data. This way, the rich client can be used as a personal information management tool, as well as an interface to shared data which is available remotely.

- *Networked Data Model:* The system's data model should provide means to associate the individual bibliographic records. It should be possible to create relationships between individual records, thus creating a network of bibliographic metadata which can be used to represent any relation between the described resources (for example, one bibliographic record could be marked as being an updated version of a publication described by another record).

- *Collaboration Support:* While the system needs to be usable in the same way as personal tools such as BibTeX and EndNote, it should also support sharing of data and other collaboration tasks. In particular, data sharing should be controllable (through user identification and access rights), and it must be possible to easily adapt the system to the rapidly changing organizational structures in typical research environments.

- *Reuse of Data:* Data should be reusable in different contexts. The core of the system is the management of bibliographic data, but this data should be easily reusable. Examples are document preparation or publication lists in HTML or for integration into a *Content Management System (CMS)* system. Data should be made available for reuse in a way that external applications can access the data without manual intervention being necessary.

Because of these requirements and the fact that the system should integrate into the existing landscape of information management applications at the local university, it was decided that an XML-based data model would be the best way to go. This way, mappings to other formats could be defined rather easily, and it could also be assured that upcoming requirements (such as the integration of new data sources or consumers) could be implemented as easy as possible.

3 XML-Centric Applications

As discussed in the introduction, XML is a grass-roots movement which has been growing from a syntax-only specification to an increasingly complex set of specifications and technologies, which are becoming increasingly abstract and powerful. However, XML has not yet gotten to the point where it has attracted enough followers to be considered a new paradigm for data modeling. While the usefulness of XML is rarely questioned, it still often is regarded as belonging to the logical model layer, and so far only little support is available when trying to do conceptual modeling with an XML-oriented data model as the final goal. Mohan and Sengupta [13] discuss this question in detail and come to the conclusion that support for conceptual modeling so far is almost non-existent.

This is a dilemma, because on the one hand there are many useful technologies for working with XML, but on the other hand there is almost no support for designing an information system that should take full advantage of these technologies. For example, Java (which we selected because of its platform-independence) includes support for many XML technologies (such as the JAXP package), but still handling XML in Java is only about handling generic XML documents, there is no support for mapping XML to language-level constructs.

Basically, when working with XML data in a programming language such as Java, there are two approaches how to handle the data: It can either be mapped to language constructs by a process outside of the programming language itself, or the language can provide native

mechanisms to handle XML data.[1] While the second approach would be preferable because it avoided hand-coded mapping and the associated problems (such as the need to cope with two models, the XML schema and the mapped programming language constructs), there currently is no established framework for this. It has been identified as a research issue and work is underway to provide support for native XML in programming languages (for example the *XJ* approach for Java described by Harren et al. [7], or the *E4X* [4] extensions for ECMAScript), but for Java this is ongoing research work.

This leaves application developers with the first approach, which is mapping XML data to programming language constructs. This can be done in two ways, one is the automated mapping by a generic framework, and the second is the manual translation of the XML schema into programming language constructs. There are several approaches for automated mapping from XML schemas to Java, the most popular is the *Java API for XML Binding (JAXB)* [6]. JAXB works for simple XML structures, but for more complex structures the mappings tend to be awkward, which is further complicated if the schema is not stable, but evolves during the development process. If the schema evolves, the JAXB mappings have to be regenerated, and this can have unwanted side-effects such as the necessity to propagate these changes in already existing code.

While schema evolution was completely out of scope for JAXB 1.0, there is some (though little) support for schema evolution in JAXB 2.0, but the fundamental problem is that there is no real framework for describing the evolution of an XML Schema. This is one of the big open research questions within the field of conceptual modeling for XML: How is it possible to describe the evolution of an XML-oriented data model without creating an entirely new model with no connections to the older version and the unchanged parts of it?

After evaluating these approaches and the associated side-effects and risks, we decided to manually map the XML Schema to Java classes, which led to a cleaner and easier understandable set of classes that the automated mapping approach. Furthermore, schema changes could be handled more gracefully, because often the changes could be implemented and propagated more locally than through a complete re-mapping of all classes, which would have been required with an automated mapping approach.

As shown in the middle of Figure 1, the resulting architecture of the system does not have a formally defined *conceptual model* (since there was no language available for this), and has two *logical models*. The XML Schema is treated as the primary model, and any changes to this model were tracked in the Java classes by manually adapting the classes to the changed schema.

As an added complication, the storage of data involved yet another mapping, because the data is stored in a relational database (which was thus used as the *physical model*). This decision was made because we wanted to support widely available database technology so that existing database systems could be used to store the data. Additionally, the requirement to support an offline mode for the rich client made it necessary to use database technology which could be integrated into a stand-alone application, and this option is only available for relational database systems (we use the Java-based HSQLDB database for local storage).

Ideally, choosing an XML database would have been better, because then only one mapping would have been necessary and the XML Schema would have provided the physical schema directly. However, the current state and availability of XML databases made it im-

[1] In essence, this is comparable to the discussion in Section 1, whether a database system is "XML-enabled" by application code doing the mapping (item 1), or by middleware provided by the database system (item 2).

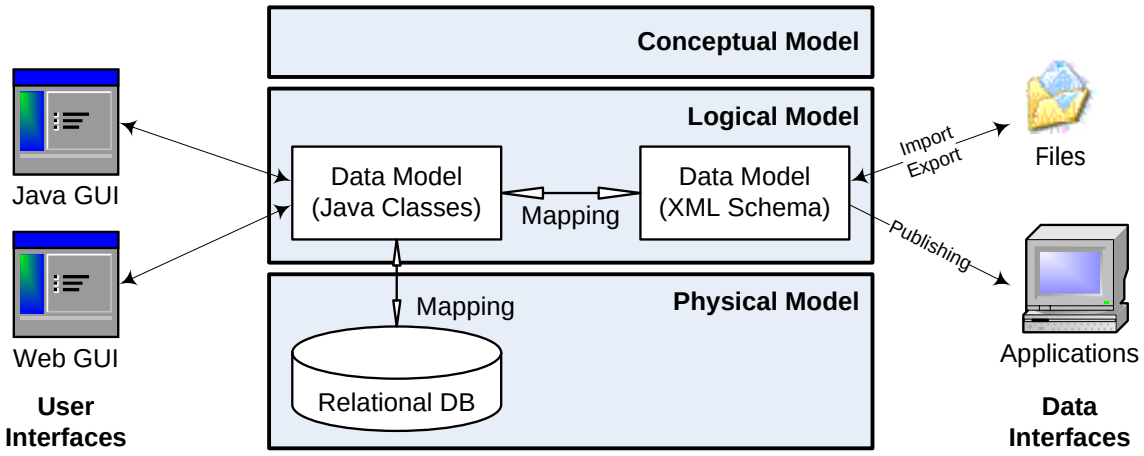

Figure 1: Data and User Interfaces

possible to find an XML database system that matched all of our requirements, thus we had to resort to relational database technology and the added effort of creating additional mappings.

Because of this design, handling schema changes proved to be a rather expensive operation, because schema changes in the XML schema first had to be mapped to the Java classes, and then to the database schema (as shown in Figure 1). Because of this cost associated with schema changes, these changes were more avoided than embraced, and this made adapting the software's data model during the development process harder than would have been ideal.

4 Available Interfaces

As described in Section 2, the application requirements were such that two user interfaces had to be supported, one as a Java-based client with a standalone option, and the other as a Web-based interface which can be accessed with a browser. Section 4.1 describes these to interfaces and the problems that were associated with aligning them as closely as possible.

As shown in Figure 1, additional interfaces were required in the form of data interfaces, which had to be provided for importing and exporting data, and for making data available to other applications. While these interfaces, which are described in more detail in Section 4.2, did not have the same requirements as the user interfaces, they still had to work on the same data model, which made it necessary to align them with the user interfaces as well.

4.1 Interactive User Interfaces

Since there were two rather different use cases, the system has to be developed with two different user interfaces. The challenge was the fact that the user interfaces should be as similar as possible (within the constraints imposed by the UI technology), but there were also some little points where the interface functionality diverged:

- *Java GUI only:* Only the Java client is able to act as a standalone client, providing access to locally stored data and synchronization features between remotely stored data and local data. Thus, support this functionality is only required in the Java GUI.

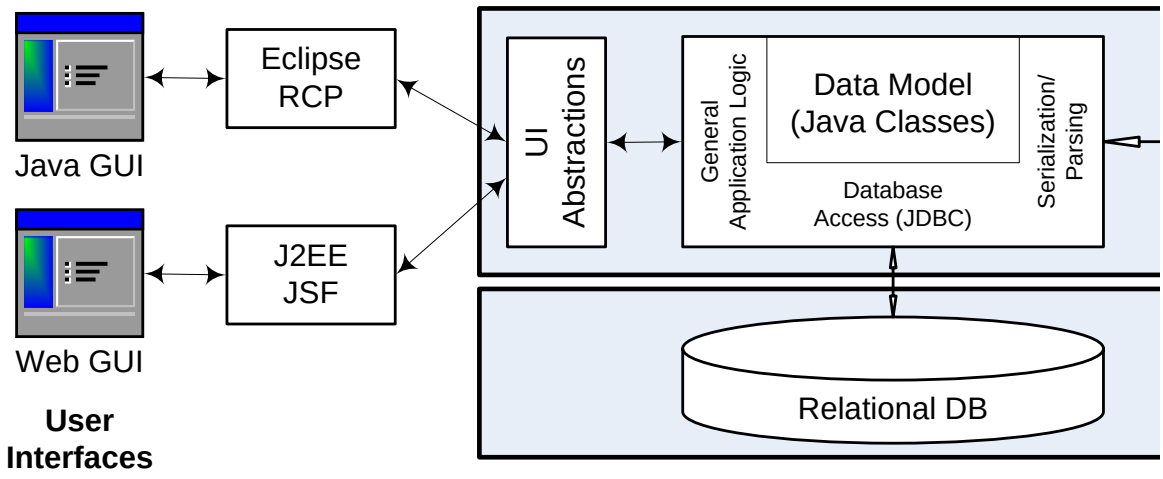

Figure 2: User Interfaces and Application and Interface Logic

- *Web GUI only:* Only the Web client allows the configuration and management of publishing channels, which are used for publishing customized views of bibliographies on the remote server. Thus, support for managing the publishing channels is only required in the Web GUI.

Apart from these minor differences, most functionality provided through the GUIs should be as similar as possible. For example, the main abstraction for working with bibliographic data are *workspaces*, which are the unit of access for users through the user interface. Consequently, workspaces are an important abstraction and are implemented so that the code can be reused for both GUIs.

Theoretically, abstract user interface toolkits such as the one described by Vanderdonckt [18] or the *Abstract User Interface Markup Language (AUIML)* by Merrick et al. [12] should support the scenario described here, providing abstraction for GUI concepts and elements which can then be mapped to different concrete implementations of GUIs. However, the existing toolkits for this kind of functionality are rather limited and did not match the requirements of the project, so the implementation was not based on such a toolkit.

As shown in Figure 2, the two GUIs implemented are a Java GUI using the Eclipse *Rich Client Platform (RCP)* and a Web-based GUI using the *Java Server Faces (JSF)* technology of the J2EE framework. Both client implementation reuse code which has been identified as providing UI abstractions. A simple example for an UI abstraction is the parsing of names when entered in a form field; this should be done consistently and is thus implemented only once an reused. A complex example for an UI abstraction is the concept of a workspace as described above.

Since neither RCP nor JSF are explicitly supporting XML as their data paradigm, some of the XML structures were hard to implement in both platforms. Any structures using traditional field-oriented structures can be implemented in both platforms rather easily, but both platforms fail when it comes to supporting mixed content. For providing mixed content editing capabilities, in both platforms there is no real support, and the amount of work that is necessary to implement mixed content editing (in particular if the mixed content is not just

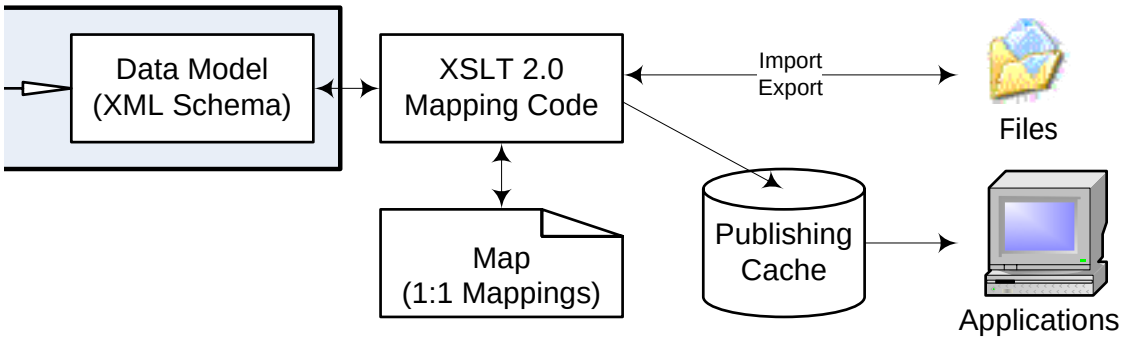

Figure 3: Data Interfaces and XSLT-based Mapping

rich text, but also may contain relationships) is substantive.

4.2 Data-Oriented Interfaces

In addition to the user interfaces described above, another important set of requirements for the system was to provide powerful support for import and export, and to provide a way for other applications to access and thus reuse data which resides inside the system.

Apart from the support of popular bibliography formats such as BibTeX and EndNote, an additional requirement was to build the system in a way which would make it easy to add support for new import or export formats. Since the system is XML-centric, *XSL Transformations (XSLT) 2.0* [10] was chosen as the language for transforming between external formats and the internal format. This way, new formats can easily be added to the system by adding new XSLT stylesheets. If a transformation is highly specialized, a user can export the internal XML format and apply his XSLT externally; if a transformation is likely to be useful for other users, it can be integrated into the system as another supports import or export format.

A recent example for this was the RFC index, an index of all Internet RFC documents, which is available in XML form and had to be made available as a bibliography inside the system. Because the RFC index is continually updated, the most reasonable way for supporting the reuse of this information inside the system was to write a new import filter.

Import and export are initiated by users through one of the GUIs. On the other hand, the system also provides access to views of bibliographies which can also be made available in the supported export formats. These views are called *publishing channels* and enable the reuse of bibliographic information. Publishing channels are configured by (1) selecting a bibliography, (2) defining search criteria within this bibliography, (3) selecting an output format for the search results, and (4) setting the formatting options for the selected output format. The results are then available at a URI at the server and can be retrieved using a simple HTTP `GET` request.

Figure 3 shown the general design of the data interfaces. One important aspect of the data interface design is that it uses declarative mappings, where a mapping table describes all data structures that can be mapped 1:1 between the internal and the external format [1]. For structures which require additional processing, code has to be written, but the mapping approach enables us to handle a large share of the required mappings declaratively, and only

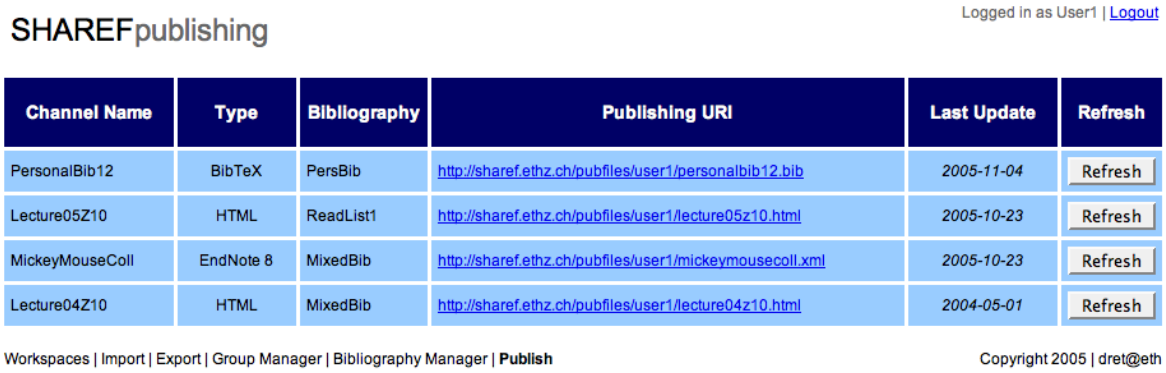

Figure 4: Web Client GUI for Publishing Channel Configuration

the special cases have to be handled by specific case.[2]

While it was an initial goal to make the software fully pluggable (so that users can plug in new import or export filters on a per-user basis), this goal was dropped because it was considered to be interesting only to a small fragment of the user population. However, making it easy to support new formats in the system was kept as a goal, so depending on user demand, new formats can be added rather easily. Because most formats have some options when being mapped, the options for each supported format are described declaratively in an XML document, and this document is used to generate the GUI for setting the options for import/export and publishing.

Figure 4 shows a screen shot of the ShaRef Web client publishing channel management. When creating new channels, the search criteria and the originating bibliography are stored for each channel. By clicking on the refresh button, the search criteria are applied on the (possibly updated) bibliography and the selected references are re-published to the publishing cache. If no manual refresh is done, the cached version is updated periodically.

The supported import formats are BibTeX, EndNote and the RFC index, and the supported output formats are BibTeX, EndNote, HTML, and Silva XML. The latter is the input format for the Silva *Content Management System (CMS)* at the local university, so that bibliography information can be imported directly into the CMS. Work is underway to enable the CMS to directly access the HTTP Web Service and retrieve the bibliographic data, at the time of writing the data still has to be exported and then manually imported in the CMS.

Apart from the RFC index and Silva XML, so far no user requests have been received to support other formats. In particular, no request for supporting *Dublin Core* [8] metadata (which is considered one of the most important metadata sets for information resource description) or any other data format has been received so far. However, the system architecture allows such requests to be satisfied easily, first by creating mappings and entering them into the declarative map, and then by implementing the mappings which require additional processing.

[2]Of course, this approach is limited by the external structures which need to be mapped. For example, when investigating the *Metadata Object Description Schema (MODS)* [11] format, the complex markup structures of this format would have made it necessary to extend the mapping mechanism, or to hand-code all mappings with XSLT code.

5 Lessons Learned

The project described in this paper was situated between pure research and simple software development. The goal was to produce a system and a service which provide useful services to researchers, but to also experiment with new technologies. Consequently, we had the liberty to not only choose established and well-known methods for software development, but to choose new technologies and build the system in a way which is not yet fully supported by existing languages and tools.

The result of the project is a usable software system build on top of established tools such as Eclipse, J2EE, and SQL. We also used many XML-related technologies, most importantly XML Schema and XSLT 2.0. During the development, the two most interesting issues were the XML-based design as described in Section 3, and the parallel design of the different interfaces as discussed in Section 4. The following list summarizes what we have learned during the project, and what similar endeavors are likely to encounter, too, given the current state of technologies:

- *Defining an XML-oriented conceptual model:* There is a noticeable gap between conceptual modeling as it should be done in information system development, and what is available for XML today. XML Schema is inappropriate for conceptual modeling and lacks essential features (such as a powerful integrity constraint mechanism), but it is the only language available today. Higher-level mechanisms such as the *CXPath* approach presented by Camillo et al. [3] may be useful for decoupling the conceptual model from concrete markup structures, but the ideal solution, a conceptual modeling language with a powerful and configurable mapping to logical XML structures, is not yet available.

- *Handling changes in the data model:* Because of the lack of an appropriate conceptual modeling language, change management becomes harder, because it becomes a manual task. This is acceptable for rather small data models such as the model of the application presented here, but should be considered very carefully for larger data models. In the manual approach we adopted, schema changes resulted in changes in the mapping to Java classes, which then resulted in the mapping to the SQL schema. Even for the rather small data model, this was too arduous to encourage an agile development process, and for any application using a bigger data model, this manual approach becomes impractical.

- *XML support in programming languages:* While generic support for working with XML documents is available in most programming languages today, in most cases there is no native support for working with data structures which correspond to XML documents (ECMAScript's E4X is a notable exception). This means that a mapping has to be performed between XML structures and the programming language's data structures. For some languages, automatic mapping mechanisms are available (JAXB for Java), but depending on the XML structures, the mapped structures can be awkward to work with. Furthermore, schema evolution is only poorly supported, and rather small changes in the schema may result in considerably more expenses for re-aligning the code.

- *Use an XML(-enabled) database:* XML support in databases and XML databases are growing in popularity and can greatly help when building XML-centric applications.

If a relational database is being used, additional mapping effort will be necessary to map the XML structures to the physical model. For small data models, this effort is acceptable, but bigger data models and the likelihood of schema evolution make it advisable to use a database system with some kind of XML support. In particular, when using document-oriented XML, the mapping to relational structures and working with these structures is awkward and should be avoided, if at all possible.

- *Use XSLT 2.0:* Even though XSLT 2.0 is in candidate recommendation status, the Saxon implementation is closely following the specification and it is very likely that no major changes will be introduced in the final phase of the standardization process. XSLT 2.0 is a major improvement over XSLT 1.0, many things can be done much easier, and other things (such as accessing non-XML files) were impossible in XSLT 1.0. However, since XSLT 2.0 is still under development, performance is not the central issue at the moment, and stylesheet execution for non-trivial code can be rather slow (this is the reason why we introduced the publishing cache).

- *Implementing multiple interfaces:* Implementing multiple interfaces requires aligning these interfaces. For user interfaces, abstract user interface toolkits would be a useful solution, but have not found their way into productive environments. When designing multiple user interfaces, common functionality should be factored out into common UI abstractions, which can then be reused from all UI implementations. If interfaces should also include data interfaces (for example, import and export), the user interfaces will probably be written in a language such as Java, while data interfaces will often used document-based formats such as XML. In this case, the alignment issue between the two data models (Java and XML) mentioned above also occur when implementing the interfaces, and every change in the data model needs to be propagated to all available interfaces, which can be cumbersome.

- *Mixed content is good, but expensive:* One of XML's big advantages over relational data is mixed content, the ability to freely mix textual content with structural information. For document-oriented applications (or applications having document-oriented parts in the data model), this can be a useful way to provide a data model that is able to capture structurally rich information. Inside the realm of XML technologies, mixed content is well supported, writing XSLT code for handling mixed content is no problem at all. In an application using components which do not natively support XML structures (such as the programming language, the database technology, the user interface toolkits), supporting mixed content can become very expensive. This added expense of handling mixed content should be taken into account when designing the data model, and only if the benefit of mixed content outweighs the additional costs of implementing it, should it be used in the data model.

To summarize, while XML is well supported in an increasing number of ways, it still is more in the status of a well-supported external data format. When designing XML-centric applications, however, it would be beneficial to have more support for XML as the internal data model of an application, and support for this kind of working with XML is not yet there.

While it is not clear that XML or one of its data models should become the main abstraction for data modeling and information system development, it is obvious that the current technologies do provide adequate support for XML-centric applications. With XML steadily

penetrating applications, it is likely that more application developers will encounter problems as the ones described in this paper, and that technologies and tools have to be developed to better support these application developers during design and implementation.

6 Discussion

This experience report reflects on the current state of technologies for the development of XML-centric applications. The overall experience of designing and developing such a system was positive, because many of the existing XML technologies and tools could be used and improved the development process.

The experience of designing and developing such a system, however, also demonstrated that there are still open issues with regard to XML support for system development. Taking today's technologies, XML is well-supported for everything that is necessary for an import and export format, and this is where XML's success started.

Because of the magnitude of XML's success, it becomes more likely that applications not only want to support it as an import or export format, but want to use it as their internal format (this is analogous to the step from XML-enabled database systems to native XML databases). For this kind of application design availability of a conceptual modeling language providing support for XML is essential, and so far there is only research work in this area.

Such a conceptual modeling language would have to be less constrained that the ER-approach and the relational model, but it would still have constraints because it probably would not support non-hierarchical concepts such as overlapping structures. What exactly such a language will look like, and whether XML will be an important part of the language itself or only of the mapping process to logical schemas remains to be seen, but the need for such a language is clearly there and will grow in the near future.

6.1 Contributions

In this paper, we have identified a number of issues which should be considered when building XML-centric applications using today's available technologies and standard software tools. These issues can serve as a general guideline for building information system which not only support XML, but should be built around XML. Since today's support for XML as the center of an information system has some critical areas, the system design and development should take these areas into account as early as possible.

As a more general result, we have identified the need for conceptual modeling languages with support for XML [20]. Today's conceptual modeling languages are mostly focusing a relational system as the target platform, and support for XML-specific features such as mixed content is non-existent on the conceptual level. While there is ongoing research in this area, it is still an open field. We are anticipating an increasing demand for this kind of conceptual modeling, because XML's success as an interchange format make it likely that an increasing number of applications would like to better integrate XML into their data model.

7 Conclusions

XML's success as an interchange format has caused many technologies and tools to be developed which today greatly easy the design and development of information system supporting

XML in some way. However, XML's heritage as an data exchange format has caused support to be largely at the perimeter of applications, providing convenient ways to generate and consume XML. When taking the next step in XML-oriented applications, which are XML-centric applications, support for application design and development is much weaker. Today, XML-centric applications can be developed based on the available technologies and tools for handling XML, but it becomes apparent that there is a need for better support of XML as the data model of applications.

The experience report we give in this paper is based on a real-world application which has been build around an XML data model. While the original goal of building an application which natively supports an XML data model could be achieved, we also experienced some problems that were not as apparent in the starting phase of the project. This report should help developers of XML-centric applications to spot potential problem areas. It also is an agenda for what is still missing in the XML landscape of today, and we hope that today's potential problem areas will all be supported by newly created technologies and tools in the next few years.

References

[1] Sai Anand and Erik Wilde. Mapping XML Instances. In *Poster Proceedings of the Fourteenth International World Wide Web Conference*, pages 888–889, Chiba, Japan, May 2005. ACM Press.

[2] 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, Candidate Recommendation CR-xquery-20051103, November 2005.

[3] Sandro Daniel Camillo, Carlos A. Heuser, and Ronaldo dos Santos Mello. Querying Heterogeneous XML Sources through a Conceptual Schema. In Il-Yeol Song, Stephen W. Liddle, Tok Wang Ling, and Peter Scheuermann, editors, *Proceedings of the 22nd International Conference on Conceptual Modeling*, volume 2813 of *Lecture Notes in Computer Science*, pages 186–199, Chicago, Illinois, October 2003. Springer-Verlag.

[4] European Computer Manufacturers Association. ECMAScript for XML (E4X) Specification. Standard ECMA-357, June 2004.

[5] Wenfei Fan and Jérôme Siméon. Integrity Constraints for XML. *Journal of Computer and System Sciences*, 66(1):254–291, February 2003.

[6] Joe Fialli and Sekhar Vajjhala. Java Architecture for XML Binding (JAXB) 2.0. Java Specification Request (JSR) 222, October 2005.

[7] Matthew Harren, Mukund Raghavachari, Oded Shmueli, Michael G. Burke, Rajesh Bordawekar, Igor Pechtchanski, and Vivek Sarkar. XJ: Facilitating XML Processing in Java. In *Proceedings of the Fourteenth International World Wide Web Conference*, pages 278–287, Chiba, Japan, May 2005. ACM Press.

[8] International Organization for Standardization. Information and Documentation — The Dublin Core Metadata Element Set. ISO 15836, November 2003.

[9] International Organization for Standardization. Information Technology — Database Languages — SQL — Part 14: XML-Related Specifications (SQL/XML). ISO/IEC 9075-14, July 2005.

[10] Michael Kay. XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, Candidate Recommendation CR-xslt20-20051103, November 2005.

[11] Library of Congress, Network Development and MARC Standards Office. Metadata Object Description Schema (MODS) Version 3.1, July 2005.

[12] 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.

[13] Sriram Mohan and Arijit Sengupta. Conceptual Modeling for XML — A Myth or a Reality? In Zongmin Ma, editor, *Database Modeling for Industrial Data Management: Emerging Technologies and Applications*. Idea Group Inc., Hershey, Pennsylvania, December 2005.

[14] Antoni Olivé. Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In Pastor and Falcão e Cunha [15], pages 1–15.

[15] Oscar Pastor and João Falcão e Cunha, editors. *Proceedings of the 16th Conference on Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, Porto, Portugal, June 2005. Springer-Verlag.

[16] Colette Rolland and Naveen Prakash. From Conceptual Modelling to Requirements Engineering. *Annals of Software Engineering*, 10(1-4):151–176, 2000.

[17] 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.

[18] Jean Vanderdonckt. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In Pastor and Falcão e Cunha [15], pages 16–31.

[19] Erik Wilde. Usage and Management of Collections of References. Technical Report TIK Report No. 194, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland, June 2004.

[20] Erik Wilde. Towards Conceptual Modeling for XML. In Rainer Eckstein and Robert Tolksdorf, editors, *Proceedings of Berliner XML Tage 2005*, pages 213–224, Berlin, Germany, September 2005.