PUBLISHED AT: IADIS WWW/INTERNET 2004 CONFERENCE PROCEEDINGS PP. 469-476

IDUX: INTERNATIONALIZATION OF DATA USING XML

Darcy G. Benoit
Acadia University
Jodrey School of Computer Science
Wolfville, Nova Scotia, Canada
Darcy.Benoit@acadiau.ca

Tomasz Müldner
Acadia University
Jodrey School of Computer Science
Wolfville, Nova Scotia, Canada
Tomasz.Muldner@acadiau.ca

ABSTRACT

Interest is rapidly growing in internationalized software that can be localized to various languages. This paper describes IDUX, an XML-based system designed to support the process of internationalizing data. A specific application of IDUX is to create internationalized websites, allowing the website owner to display information (such as his or her resume) in multiple languages. To support the reusability of translations, data and their translations are persistently stored in databases, using a novel approach to interfacing databases and XML data conformant to multiple XML schemas.

KEYWORDS

Internationalization, XML, databases

1. Introduction

In the past, most software programs could only "*speak one language*". For example, software developed in the UK could speak English, while the same software developed in China could only speak Chinese. (In this paper, a *language* refers to the natural language used in the **H**uman-**C**omputer **I**nterface (**HCI**). Unless it is clear from the context, we always say "a programming language" when we refer to a language used for programming.) Therefore, multiple versions of a single program that differ only in HCI language might require completely different implementations, resulting in error-prone duplications of the original code. This situation was unacceptable for two reasons. Firstly, multi-language countries (such as Canada) may necessitate that some software be produced in each official language (English and French). Secondly, growing globalization often results in products which are developed in one country and shipped to several other countries. This globalization forced software developers to rethink the software development process and tackle the issue of internationalization. *Internationalization* of a product means that the product can be adapted to various languages without making any changes to the architecture. The internationalization of a calendar would result in the month and day names being displayed in the appropriate language. *Localization* of the internationalized product refers to the adaptation of this product to a specific *locale*, which describes the language. An internationalized Java

calendar that has been localized to French will show dates in the format used in France. Due to the length of the terms *internationalization* and *localization*, the short, mnemonic terms of I18N and L10N are used respectively.

Product internationalization and localization is difficult because it goes beyond simple functional suitability of a program and considers the many facets of HCI language, such as alphabets and scripts, spacing rules, text direction, date and currency formats, sort orders, etc. Despite the difficulties, internationalization interest is growing rapidly with more applications being internationalized regularly [12]. Many companies, such as EXCEL Translations [8] specialize in internationalizing existing applications.

In [19], we described a system called **I**nternationalized **F**aculty **W**ebsite (**IFW**), which can be used to create an internationalized website showing the Curriculum Vitae (CV) for a faculty member. A user of IFW (the *creator* of the webpage) enters his or her data by following a series of GUIs in a host language such as English, and then forwards the project to the IFW *administrator*. The administrator is responsible for finding *translators* for the task and *verifiers* who check the completed translation. The final product, returned by the administrator to the creator, is a website that will initially be displayed in one language but will have the option to be displayed in a variety of languages chosen by the creator. In addition, the creator is able to specify one or more selections of data to be shown (for example, only journal publications), and one or more of selections of formats these data are to be rendered in (such as HTML or PDF).

Internally, CV data used by IFW are stored in XML, and an XML document representing CV data for a single faculty member is built to conform to a *single* XML schema that defines a specific kind of CV. In this paper, we generalize the approach taken to designing IFW and describe IDUX; a system for internationalizing data that can conform to multiple XML schemas. A specific problem that had to be tackled in designing IDUX was a support for persistent storage of XML data representing various translations that are conformant to different XML schema. XML-enabled relational databases often provide interfaces to manually or automatically create tables representing a single XML schema, but with multiple XML schemas, this process will result in a large number of tables and therefore will be inefficient.

In this paper, we describe a novel approach to the problem of storing in relational databases XML documents that conform to multiple XML schemas, which limits the number of tables and makes database operations efficient. We briefly describe the functionality of IDUX and concentrate on the implementation issues. For implementation, we use various recently developed software tools based on Java and XML, such as JAXB, versioning of XML documents, and relational databases with XML support.

This paper is organized as follows. Section 2 covers related work. Section 3 explains our approach to internationalization using XML, and describes how multiple XML documents can be efficiently stored in a relational database. Section 4 concludes the document.

2. Related Work

In this section, we briefly describe the support for the internationalization given by XML and we outline a process of automatic translation; for more information on the internationalization process, see [23]. There are many advantages of using XML [37]data for internationalization:

1) support for Unicode
2) Separation Of Concerns (SOC): describe content rather than formatting
3) ease of converting to various formats, including HTML, VoiceXML, etc.
4) support for mixing several languages
5) help to avoid repetitions by storing in, and retrieving from, databases
6) support for specifying translatable text

Data-centric documents are documents that use XML as a data transport, such as sales orders. *Document-centric* documents are documents that do not have a very regular structure, such as email. A database is *XML-enabled* if it allows the user to store and retrieve XML. XML-enabled relational databases are not specifically designed to store XML data and thus require some form of middleware to map the XML data to the relational tables. This is

done by mapping the XML document schema (DTD, XML Schemas, RELAX NG, etc.) to the database schema. The data transfer software is then built on top of this mapping, (for details of this approach, see [2].

Mainstream database management systems (DBMSs) already support some type of mapping between XML and relational data. These methods usually involve creating a mapping between the XML schema and the relational schema. Tools are provided by the DBMS vendors to ease this particular task. Once the mapping has been determined, it is possible to insert, retrieve and update the XML data in the relational database. In keeping with their experiences, the mainstream DBMS vendors have requested that XML support be added to the next version of SQL. SQL/XML (or SQLX, as some refer to it) allows users to form SQL queries that create XML structures and to specify how relational data is to be converted to and from XML [3].

Computer-assisted translation uses **T**ranslation **M**emory (**TM**) systems that typically consist of a translator module, an editor module, and a database of terms. TM software stores language segments translated by translator in a database for future reuse. Translators working on text segments can invoke fuzzy searches for these segments and use the results retrieved from the database. Some well known companies offering TM systems are Déjà Vu [5], the Translator's Workbench from Trados [29], and the STAR Transit [30]. TM software typically uses the **T**ranslation **M**emory e**X**change (**TMX**) format, which is a standardized XML document type for storing collections of segments in multiple languages. For more information on TMX, see [16].

Using TM software for translations has both advantages and drawbacks. Firstly, TM software views the source text as a collection of text units called *segments*. Segments may range in size from simple text strings to paragraphs. The technique used to break up the text into segments is called *segmentation*. Segmentation may remove the context in which the text segment appeared, resulting in an incorrect translation. An example in [26] shows that the English word “Help” translates to two different French words depending on the context in which the word appears. The common solution to the context problem is to use a verification phase in which the translator reads, verifies and possibly corrects the translation. The second problem with TM systems is that they are expensive, both in terms of the price of the software and the need to hire specialized personal able to use these systems. (More on automatic translation in [7]).

3. Internationalization of Data Using XML

3.1 Introduction

This section describes the process of internationalizing data using IDUX. In this paper, we briefly describe the functionality of IDUX, and then concentrate on its implementation.

3.2 Users of IDUX

The design of IDUX uses **s**eparation **o**f **c**oncerns (**SOC**) to separate tasks that require different type of technical expertise. There are four kinds of IDUX users:

- creators (specify the XML schema and enter XML data conforming to this schema)
- administrators (maintain accounts, XML schemas, forward documents, etc.)
- translators (translate documents submitted by administrators)
- verifiers (verify translations submitted by administrators)

IDUX is a distributed system, consisting of a central server (IDUX server) and users accessing the IDUX server through the Internet. Any browser can be used to access the IDUX system. Users must have accounts on the IDUX server. Administrators of the IDUX server create, modify and delete accounts for all users and maintain repositories of available translators and verifiers. In general, internationalized data created by IDUX can be used in any document; in this paper, we describe documents that are websites.

Initially, the creator sends a request to the administrator to create an account. Once this account has been made available, the creator can select one of the existing XML schema, for example the one that defines a faculty CV, or the creator forwards to the administrator a request to use a new XML schema. Note that the creator does not have to be fluent in XML; the only time he or she has to explicitly deal with XML is when specifying the schema. In the next step, the administrator uses the software which creates a GUI that can be used to enter data that are grammatically correct according to the given XML schema.

There are several kinds of languages used in IDUX:

- The *interface language* is the HCI language that appears in IDUX's GUIs. The current version of our system uses English as an interface language.
- The *source language* is the language used by the creator to enter data
- The *primary language* is the default language selected for the final product. For example, if the primary language is Spanish, then initial access to the website will be in Spanish. The website will also be available in the other languages selected by the creator when using the IFW system.
- The *secondary language* is any language specified by the creator as one of the languages the final product can be displayed in (therefore the data have to be translated from the source language to the primary language, and all the secondary languages).

After the creator entered her or his data, they select various options to affect the creation of the final product. These options include the selection of:

- the primary display language for the website
- one or more secondary languages available for display
- data to be shown (all data, only journal publications, etc.)
- formats in which data can be rendered (HTML, PDF, PostScript, etc.).

The workflow for creating internationalized data can be summarized as follows:

1. The creator asks the administrator to create an account for her or him
2. The creator uses the account to access the IDUX server and selects an existing XML schema or forwards a new schema to the administrator
3. The creator selects the source language and enters her or his data, using a GUI provided by the administrator
4. The creator selects the primary and secondary languages
5. The administrator submits translations tasks to translators
6. When any translation is completed, it is returned to the administrator who forwards it to the verifier
7. Once all translations are verified, the creator is notified

The administrator may reject a request to create an internationalized website if there are no available translators or verifiers for the requested translations. Also, the administrator may reject a request if the tool to produce the required rendering is not available (for example, a request to create an RTF format). However, the creator may suggest to the administrator the availability of such a tool that the administrator can add to the IDUX server. Various administrative tasks may be automated, such as assigning submissions to translators. The translated text is permanently stored in the IDUX server with no repetitions, allowing for reuse as needed. The owner is able to modify data and resubmit them for translation.

3.3 Final Product and Maintenance of Internationalized Data

In the previous section, we described the process of creating an internationalized data. This section provides a more detailed description of what the final product – an internationalized website – looks like, and what the specific requirements are on the system serving this website (besides the standard requirement of a web server).

The final product is a website, containing one default webpage (displayed using a primary language), which can be used to:

- choose a secondary language to display CV data
- choose which data will be displayed
- choose a format for display.

The final product may appear in one of two available kinds:

- transient. All translations are stored in the database, and webpages accessible from the default page are dynamic. The website has to provide the same functionality as the IDUX server; see Section 3.4.3.
- persistent. All webpages are static, generated by retrieving data from the database, and applying all transformations.

From the perspective of the client, who is accessing the website, there is no difference in what kind of final product is used. However, choosing a specific kind influences requirements on the server side, and efficiency of the website.

The generation process is performed by the IDUX administrator. This process is time consuming and therefore should be performed only if the CV data are not to be frequently changed. On the other hand, a persistent product, consisting entirely of static webpages, reduces the overhead caused by creating dynamic webpages, and does not impose any additional requirements on the website server. Therefore, the persistent product can be copied from the IDUX server to any site with the standard web server. A transient product requires a website with the server that can handle servlets, access a database, etc.

Maintenance of CV data, such as adding a new publication, requires a submission of the request to the administrator. Once translations of new data are preformed and verified, the transient product is available. However, if semi-persistent or persistent requested, then the administrator will generate this product.

3.4 Implementation

IDUX is implemented with the help of several recently developed software tools using Java and XML; specifically, JAXB, databases (either native XML or relational that support XML), and versioning of XML documents. This section briefly describes the implementation; for a more details, see [19]. Internally, all data are stored in XML. Specifically, we use the XML File Interchange Format (XLIFF) [35]. The advantage of the latter technique is that it is database-independent and there are various translations tools available that use XLIFF. Note that it is easy to convert XML to XLIFF using XSLT stylesheets [26]. The XLIFF format looks roughly like this:

```
<trans-unit>
   <target> … </target>
   <target> … </target>
</trans-unit>
```

The creator enters data by working as an IDUX client (see below), and these data are stored in the database on the IDUX server. To implement the programs used by all kinds of users, such as creators or translators, we use JAXB [22], which is a Java technology that makes it easy to read, modify and write back XML data. JAXB hides from the internal XLIFF representation of data and shows appropriate GUIs to create data, translate them, etc. The text extracted from the database is handed over to one or more translators, who will be provided with the text, which is extracted from the translation fields (values of the target elements in the above example).

3.4.1 Databases

The current implementation of IDUX uses DB2 from IBM, but our approach is applicable to any relational databases; no matter whether or not it is XML-enabled. IDUX will access the database to perform several of its basic tasks, such as:

- building a localized XML document (such as a CV) for a specific language

- maintenance of the database, such as adding new faculty members, removing and updating data for existing faculty members
- performing translations and verification tasks.

It may look attractive to use the XML extender for DB2, which makes it possible to write queries that return XML documents. However, the use of the extender requires a new **D**ocument **A**ccess **D**efinition (**DAD**) file for each XML schema. DAD files are created to map individual schema to the relational model [14]. This approach has many downfalls. One such problem stems from the need to create a new DAD file for *each* new XML schema (recall that IDUX can accept document conforming to multiple XML schemas). Given the possibility of hundreds of different document structures, it is not efficient to create a new DAD for each structure. A second problem associated with creating a DAD file for each different XML schema involves the number of relational tables involved. DAD files are used to "translate" an XML document into a defined relational schema. A new relational table would need to be created for each new XML schema, resulting in the DBMS needing to maintain numerous tables. With the possibility of large number of XML schemas, the requirement that a new relational table be created to manage a single XML schema is not efficient.

In order to overcome the issue of requiring a new relational table for each XML schema, we propose a new approach that would use only two tables; one table to store document structure and a second table to store the data associated with each leaf node. This solution allows us to store many different XML documents conforming to various XML schemas, without having to create a new DAD or relational table for each.

Database Structure

The basic idea of our approach is borrowed from the implementation of a forest in two arrays, using a left-most-child/right-sibling tree storage format [1]. Therefore, our proposed method for the storage of generic XML documents in a relational database requires two relational tables.

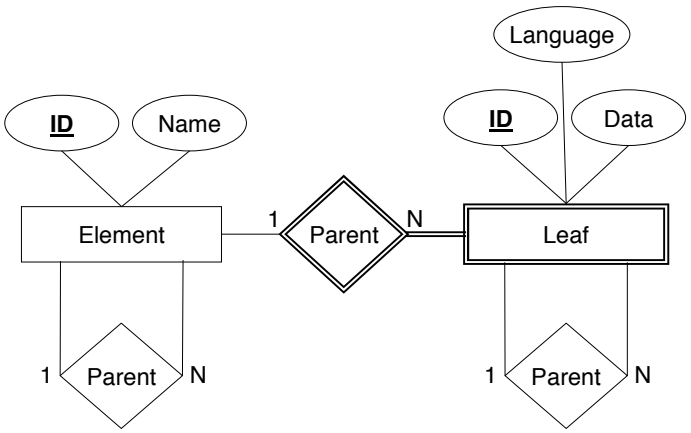

Figure 1 is a partial Entity-Relationship (ER) diagram of the proposed entities. It should be noted that several relationships with management entities have been left out of this model for simplicity. Our two main tables are the "element" and "leaf" tables. The element table will be responsible for storing the structure of the document. Using a left-most-child/right-sibling tree storage format, the tree structure from the original XML document is stored in the element table. Each individual tree is uniquely identified and associated with the user who submitted the document into the system. Element data is not stored in the element table, but the leaf table. Individual instances of an element are stored as tuples in the leaf table and are directly related to their parent element. This structure allows for multiple leaf nodes to be associated with an individual element entry.

Consider, for example, a user submitting a document with a short biography to the system. Assuming that the biography is treated as a single element within the document, the structure of the document will be captured with

one of the elements being a biography. The text of the biography will be stored in the leaf table, with the entry directly related to the element table. The rest of the data in the document will also be stored in the leaf nodes.

Each leaf node is required to record the language, in which the data is stored. As translations of the original leaf node data are inserted into the database, they will be entered as new leaf nodes in the node table. When inserted into the table, the ID value for the leaf node of the original text will be stored as a part of the new tuple. The ID number from the element will also be included in the translated tuple. This information will allow the system to search the leaf table for all data associated with an individual element associated with a specific document. This data may be stored multiple times in the leaf table, with each tuple being an instance of the same data in a different language. The child-parent relationship stored in the leaf table allows the system to record the source data for each translation. This allows the system to reconstruct the translation tree for any given piece of data, allowing the user to view the source data and the translated data. This translation information is useful in cases where intermediate languages are used during the translation process. If the entry for the intermediate language is modified in the database, then the translations of that data must also be modified.

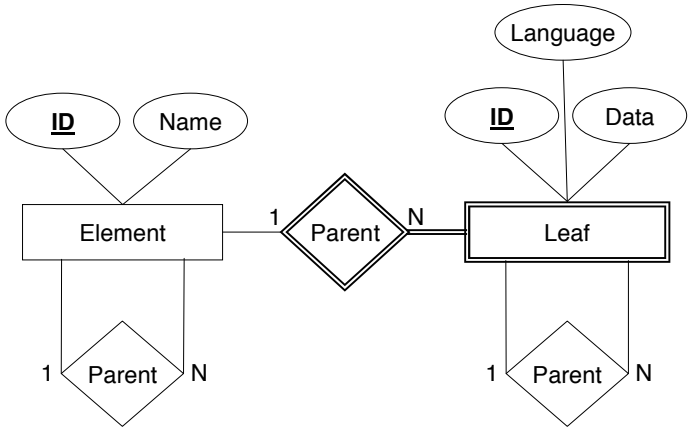

Figure 1 - ER diagram for the Element and Leaf entities.

Concerns

Several concerns must be addressed when storing XML documents in databases [2]. The first has to do with the issue of mapping document schemas to the relational database model. As mentioned earlier in this section, we avoid a strict relational mapping from an XML Schema to the relational model, because our relational model must be able to handle multiple XML Schemas. A related issue is the ability to generate an XML Schema from the relational schema. IDUX has no requirement for this ability, as we are only concerned with rebuilding individual documents. A second issue concerns the ability to store schema-less documents. As we mentioned earlier, for every XML schema, there is a GUI used by the creator to enter data, and therefore XML documents stored in the database are always valid according to some schema. There is no need to store schemas in the database, because XML documents do not have to be revalidated. A third issue associated with storing XML documents in a relational database is the ability to query the database and return XML. Querying is done in the same way as any relational database, and then the data returned are translated into XML. A final problem is that of dealing with "round-tripping" XML documents, which is the ability to insert a document into the database and be able to retrieve the exact same document. Since IDUX is mainly concerned with the elements of the document, the ability to "round-trip" a document is not required.

3.4.2 IDUX Client and Server

An IDUX client can use any web browser. The IDUX server requires the following functionality: handling servlets, access to a database, JDK 1.4, and Java Web Services Development Pack, WSDP.

4. Conclusions and Future Work

This paper described the design and implementation of an IDUX system that can create internationalized data. IDUX can be used to create websites that "can speak many languages". We proposed a new approach to efficient storage of schema-less XML documents in relational databases.

Since the current state of automated translation systems requires human intervention, this version of IDUX uses no automated translation. However, its design, in particular the use of XLIFF, makes it possible to include future versions of automated translation systems with no major changes of the IDUX architecture.

Future work includes internationalizing the IDUX system. Once the initial system is created in a single interface language, the system will be used on itself in order to generate new versions of the interface. IDUX data will always be kept up-to-date with any translators that may be available to the system. As new translators are added and new languages are added, the interface will be translated to represent the new languages available for translation.

In addition, our future work involves defining a common interface that will be used by the implementation of IDUX to talk to any database, and will make IDUX's design database-independent. With this design, we will be able to plug-in different databases, including pure relational databases, XML-enabled databases (such as Microsoft SQL Server [17] that uses a schema language XML Reduced (XDR)), and native XML databases (such as Xindice [34]). We will also experiment with XML versioning systems [6] to compare new translations and accepted translations to retrieve parts that have to be corrected. Finally, we are currently working on implementation of algorithms that automatically select translators and verifiers, releasing administrators from the task of assigning these duties.

References

[1] Aho, A. V., Hopcroft, J.E., and Ullman, J. D., *Data Structures and Algorithms*. Addison-Wesley, Mass, 1983.

[2] Bourret, R. (2003) XML and Databases, http://www.rpbourret.com/xml/XMLAndDatabases.htm

[3] DataDirect Technologies (2003). SQL/XML in JDBC Applications, http://www.datadirect-technologies.com/products/connectsqlxml/docs/sqlxml_whitep.pdf

[4] Deitsch, A., & Czarnecki, D. (2001). *Java Internationalization*. O'Reilly

[5] Déjà Vu (2003) http://www.atril.com/

[6] Delta XML (2003) http://www.deltaxml.com/

[7] Dennett, G., (1995). Translation Memory: Concept, products, impact and prospects. South Bank University http://www.star-uk.co.uk/About_us/People/Gerald_Dennett/msc.pdf

[8] EXCEL Translations (2003) http://www.xltrans.com/ser_tra.html

[9] Fitzgerald, M. (2003) *Learning XSLT*. O'Reilly

[10] Google (2003) http://www.google.com

[11] Hall, M. (2001) *Core Servlets and Java Server Pages*. Sun Microsystems Press/Prentice Hall PTR.

[12] HRS (2003) http://www.hrs.de/

[13] Itagaki, M. (2000) Use XML as a Java Localization Solution. http://www.fawcette.com/Archives/premier/mgznarch/xml/2000/05win00/mi0005/mi0005.asp

[14] IBM, DB2 XML Extender webpage http://www-3.ibm.com/software/data/db2/extenders/xmlext/

[15] Java (2002) http://java.sun.com/j2se/1.3/docs/guide/intl/index.html

[16] LISA (2003). The Localization Industry Standards Association. http://www.lisa.org/tmx/

[17] Microsoft SQL Server (2003), http://www.microsoft.com/

[18] Mozilla project (1998) http://www.mozilla.org/docs/refList/i18n/

[19] Müldner, T., Wong, F. and Benoit, D. (2003). Internationalization of Websites. *TR 2003-04*, Acadia University, 2003

[20] NetBeans (2003). http://www.netbeans.org/

[21] OmniFormat (2003) http://www.omniformat.com/

[22] Ort, E. & Mehta, B. (2003) Java Architecture for XML Binding (JAXB) http://developer.java.sun.com/developer/technicalArticles/WebServices/jaxb/

[23] Raetzmann, M. & de Young, C. (2003) Galileo Computing Software Testing and Internationalization. TeriR@lemoine-international.com

[24] Rajgopalan, S. (2003) Software Internationalization: A Holistic View. Advisor. http://portalsadvisor.com/doc/12841

[25] RWS (2003) http://www.translate.com/locales/en-US/companyinfo.html

[26] Savourel, Y. (2001). *XML Internationalization and Localization*. SAMS.

[27] Seshadri, G. (2000) Internationalize JSP-based Websites. *Java World*, http://www.javaworld.com/javaworld/jw-03-2000/jw-03-ssj-jsp.html

[28] SUN (2002). Internationalization. http://java.sun.com/j2se/1.3/docs/guide/intl/index.html

[29] Trados (2003) http://www.trados.com/

[30] Transit (2003) http://www.star-ag.ch/eng/software/sprachtech/transit.html

[31] Unicode (2003) http://www.unicode.org/

[32] Webmail (2003) http://www.webmail.co.za

[33] WordLingo (2003) http://www.wordlingo.com/

[34] Xindice (2003) http://xml.apache.org/xindice/

[35] XLIFF (2003). XLIFF 1 Specification. http://www.oasis-open.org/committees/xliff/documents/xliff-specification.htm

[36] XML (2003) http://www.wordlingo.com/

[37] XML FAQ (2003) XML Internationalization and Localization FAQ. http://www.opentag.com/xmli18nfaq.htm

[38] XPath (2003) http://www.w3.org/TR/xpath