USING COCOON TO BUILD GLOBALIZED WEBPAGES

Tomasz Müldner, Zhinan Shen and Li Bo Ya
Jodrey School of Computer Science, Acadia University
Wolfville, NS, Canada B4P 2R6

ABSTRACT

Globalization systems are designed to internationalize webpages by making them available in various languages. In this paper, we describe design and implementation of a system called CGF (Cocoon-based Globalization Framework) to build globalized webpages, based on a popular Apache Cocoon web development framework. CGF supports users in various roles, including creators of webpages and text translators, and separates these roles to make the translation process transparent to the creator. Therefore, users without any programming experience can create data in one or more languages, request translations, and then receive webpages available in requested languages.

KEYWORDS

Internet technology, internationalization, web pages

1. INTRODUCTION

The global nature and accessibility of the Internet has generated interest in globalization, i.e. making documents such as websites available in various languages. Globalization consists of two phases: internationalization and localization (Savourel, 2001). Internationalization entails generalizing the product so that, without the need for redesign, it can be localized to specific languages and cultural conventions. Due to the length of the terms internationalization and localization, the short, mnemonic terms of I18N and L10N are used respectively. (The shortened terms are names based on the number of letters between the first and last letter of each word.) There are many existing globalized systems such as the Hotel Reservation System, see HRS (2003), where the user can choose one of 31 available languages, and Webmail, see Webmail (2003), which is a popular email client running from any browser, whose user can choose one of over 28 languages.

A more precise definition of internationalization, borrowed from Belussi and Posenato (2004) states that '*...the internationalization of web applications regards the implementation of a mechanism that is able to produce, for each page the web application generates (master page), a set of pages with the same content but each one in a different language (translated pages)...*'. The system described in this paper satisfies this definition and supports *automatic generation* of multi-language webpages.

The Cocoon-based Globalization Framework (CGF) can be used to build globalized webpages *from scratch* rather than globalizing existing webpages. These webpages use semi-structured source data that adhere to so-called *data definitions*, such as the Course Description data definition. CGF provides numerous data definitions that can be used and optionally customized by users, for example the Course Description data definition may have some of its elements removed, or new elements be added. It should be noted that to accommodate various needs, completely new data definitions can be added to CGF.

The creator of a globalized webpage selects the desired data definition, enters source data to populate the website, specifies target languages, in which the webpage will be localized, and then submits a request to perform translations. The latter task is performed by different kind of a user, called a *translator* who can not modify data or add new data but will translate these source data to other target languages. Of course, translations should be *reusable*, using some form of the Translation Memory (TM), which looks up the source and target (translated) phrases in the persistent storage to help in the translation process. CGF supports TM and it also supports re-translation of the modified source data so that the previous translations of other data are left intact. Once all translations are completed and verified (for details, see Section 4.1.3), the creator receives all translated data. At this point, the creator can specify a specific format in which these data may be rendered, for example, HTML or PDF, and generate the corresponding internationalized webpages.

The implementation of CGF is based on a popular Apache Cocoon web development framework (1998). There are many reasons why Cocoon is an ideal choice for our system. First, it supports Separation of Concerns, SoC, which is an essential feature of CGF. Second, Cocoon is based on components, from which pipelines to perform required tasks can be created. We built various specialized components, and we are taking advantage of flexibility offered by pipelines; for example to provide email notifications sent to creators when their translation requests are complete.

This paper is organized as follows. In Section 2, we review the related work; in particular, we describe our previous work on website globalization and contributions provided in this paper. Section 3 briefly describes Cocoon, and Section 4 talks about the design and implementation of CGF, and provides screenshots showing how this system works. Finally, Section 5 provides conclusions and describes our future work.

2. RELARED WORK AND OUR CONTRIBUTIONS

There are several interesting projects on globalization; for example, Aykin (1999) suggests rules for website internationalization, and Belussi and Posenato (2004) propose a framework for the internationalization of existing websites, which are *data-intensive*, i.e. using dynamic web pages generated by data stored in relational databases. The framework has been used at the University of Verona to provide a website in five languages. The main idea of this framework is to use database extensions to store translations and query rewriting to provide translations. Therefore, the framework is limited to the websites that are data-intensive in the sense described above. In addition, it does not provide a structured description of the translation process and translation reusability. Finally, Yu (2003) describes the application of XSLT to localize XML documents; however, this work does not attempt to develop a uniform approach to globalization.

Our initial research (Benoit and Müldner, 2004; Müldner and Benoit, 2004; Müldner et al, 2004) concentrated on applications of various XML technologies for globalization, including XML Schema (World Wide Web Consortium, 1994), and XSLT (World Wide Web Consortium, 1994). We designed and implemented the Internationalized Faculty Website system (Müldner et al, 2004), which can be used to create globalized products containing the Curriculum Vitae (CV) for a faculty member. We also designed and implemented a system for the Internationalization of Data using XML, IDUX (Benoit and Müldner, 2004), in which we extended the original IFW system by allowing the user to use a variety of data definitions (and not just CV data). We implemented a preliminary version of this system using a relational database to store XML data (Müldner and Shen, 2005). Our experiments with these systems revealed various weaknesses of the design and implementation.

Main contributions of this paper include a description of the re-designed and re-implemented globalization system CGF, which shares some features with its predecessors described above. However, unlike previously built systems, CGF is a *Cocoon-based*, modular, and flexible system that can be easily extended by adding additional components and pipelines, and it can be modified by changing existing components and pipelines. For example, to experiment with the translator component, one can create several versions of this component, and then replace one component by another, without making any changes in other parts of the system. CGF provides persistent storage for translations by communicating directly with a selected native XML database (currently we use Exist), which turned out to be a much better choice than an indirect communication with a relational database. The second contribution is the creation of a complete globalization system that can be used by naïve users to build webpages in many languages.

3. MAIN FEATURES OF COCOON

In this section, we summarize main features of Cocoon and focus of its applicability for implementing globalization systems. Cocoon (1998) is a framework for building XML-based dynamic websites, which supports the following features:

- clean separation of content, logic and style
- server-side on-the-fly XML processing
- XML trees represented as linear streams of events

Cocoon also supports *content syndication*: parts of a web page may come from various sources. This is essential for I18N applications, because instead of having multiple copies (in various languages) you can have a single copy and an XSLT stylesheet which performs translation using a message catalog, which is Cocoon's term for translation memory.

One of the most important concepts in Cocoon is that of a *pipeline*, which is a sequence of one or more components; each of which (except the first one) reads input from its predecessor, and each of which (except the last one) writes its output to its successor. There are various types of *components*; including:

- *generators* which read data sources, such as XML documents, directors, and images and possibly converting them to XML
- *transformers*, which read events from generators, and convert its input (possibly with side-effects such as sending email), using XSLT; for example to create specific formats such as PDF
- *serializers*, which produce a stream of bytes, and are used as the last component in a pipeline

The power and flexibility of Cocoon comes from the fact that a programmer can use existing components, build new components, and create pipelines of components. Components can be configured, using parameters, and they are described in the file sitemap.xmap, a control center, which also describes the configuration of pipelines. To make management of a sitemap more manageable, it may consist of a number of sub-sitemaps that can be mounted in the main sitemap.

Cocoon's latest version comes with an I18N transformer, which is built specifically for I18N and L10N applications, and supports the following features:

- Text translation
- Attribute translation
- Parameter substitution (with translation if needed)
- Date, number and currency formatting

4. COCOON-BASED GLOBALIZATION FREMEWORK

We start this section by describing the functionality of CGF and showing various screenshots of this system in action. Next, we describe the implementation, focusing our discussion on the use of Cocoon.

4.1 Functionality of CGF

4.1.1 Introduction

There are five kinds of user **roles** in CGF:

- creators (enter source data, select languages, etc.)
- administrators (maintain accounts, etc.)
- translators (translate submitted documents)
- verifiers (verify submitted translations)
- end-users, who access internationalized webpages

A user can create her or his account, but this account will not be active until it is approved by the administrator. Accounts are needed for the first four roles described above. A creator builds a translation task, submits it, and when the translations are completed, they are made available to this creator. The reason that we provide a *verifier role* is that the translator may or may not see the context for the complete translation. Therefore, there is a need for the verification of the translation.

CGF uses semi-structured source data that adhere to data definitions (see Section 4.1.2), such as the Book data definition, for a book with chapters, sections, and pages. CGF provides numerous data definitions, such as Curriculum Vitae (CV) data definition, Courses data definition, etc., and the creator selects the appropriate one (or creates a new data definition).

CGF maintains two kinds of general *repositories* of various resources; in particular, data definition resources. A global repository is maintained by the administrator and accessible to any creator. A local repository is maintained and accessible only to a specific creator. Therefore, the administrator is responsible for the quality of a global repository, and the creator is responsible for their local repository. Resources in

both repositories may be classified by creating a hierarchical tree of folders; for example all data definitions can be stored in a Data Definition folder, which may have nested subfolders.

4.1.2 Creators

In this section, we describe actions performed by creators.

Data Definitions and Translation Tasks

The creator first logs into CGF, and selects a specific data definition. In this section, we show screenshots based on the Book data definition, here called B-DD; the actions executed by the creator for other data definitions are identical. For a selected data definition, the creator can create multiple webpages showing different content; for example for two different books. To help the creators organize their work, CGF allows them to place translation tasks into *packages*. Therefore, the creator first creates a package, which will store one or more *translation tasks*. Each Translation task (TT) has the following attributes: a name of the TT, a name of a creator, date when this TT has been submitted (if any), the status, a short description (to help the user identify the purpose of the TT), and one source language and one target language. A TT can be in one of the following states: "not submitted", "submitted", "assigned for translation", "in translation", "translated", "in verification, "rejected", and "completed". A TT is initially in the "not submitted" state, in which the creator can modify its data. After the TT has been submitted (sent for translation), it may be in one of other states listed above and it can not be modified in any of these states. (These other states will be explained in the following sections.)

Note that each TT has one source and one target language, and if the creator wishes to provide multiple target languages, she or he can create additional TTs, each of which would have a different target language. The creator does not have to enter source data multiple times; it is enough to do it for one TT and then use these data for other TTs.

Now, let's describe how the creator enters source data. A data definition is presented to the user in a form of a GUI, called a *Data Definition Form*, which allows the user to enter only data valid under this definition. A data definition form B_DD that uses English as an interface language is provided in Fig. 1a).

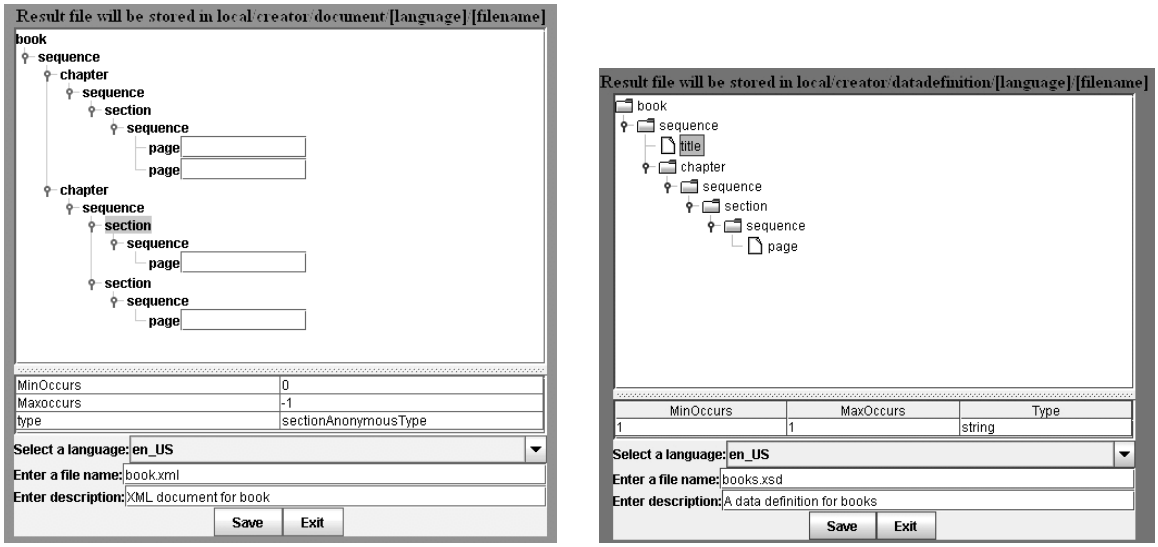

Figure 1. a) Data definition form for B-DD b) Modification of B-DD

CGF allows the creator to modify the existing data definition and store in a local repository. Examples of modifications are adding or removing a specific field. These modifications can take place at two levels. First, at a higher level, a data definition may be modified, effectively creating a new data definition. Second, at a lower level, a data definition form may be modified, to accommodate a single-use creation of a website. Fig. 1b) shows the process of modifying B-DD by adding the title of the book.

Various fields in the data definition form presented in Fig. 1 have *labels* in a certain language, which is referred to as the *interface language*. A preferred situation is the one in which the interface language is the same as the language used for the source data; however, if this interface language is not available, it may be enough for the creator to use a language for which they can *understand* labels. A data definition form is automatically *generated* by CGF, and it may be selected by the creator from one or more available interface languages to allow the entry of data in the creator's preferred language. If the required interface language for the data definition form is not available, the creator can request to translate labels in form, rather than the source data.

The creator may know more than one language, and in such a case she or he will not have to submit a TT for translation. In this case, the creator builds a TT with identical source and target language. It is the creator's responsibility to enter data, which are grammatically and semantically correct according to the selected *source language,* and the creator does not send this data for verification. For example, assume that the creator knows English and Polish, and uses a data definition form for B-DD with English as the interface language. She wishes to create a webpage that can be localized to English, Polish and Chinese. Therefore, she creates a package and then within this package she creates two TTs, one with English as the source and the target language and another one with Polish as the source and the target language. The creator then enters data for these two tasks. These two tasks will not be sent for translation. As a result of the creator-provided data in English and Polish, there will be two *products* available. Then, the creator builds another TT to specify a required translation, for example with Polish as a source language and Chinese as a target language. The exact choice of languages for the last two translation tasks depends on availability of translator; for example if there are only translators available from English to Chinese, then the last TT would have to use these languages.

Once the translation request is submitted, the TT state changes; first to "submitted", then when the TT is given to a translator to "assigned for translation". Once the translator starts working on the translation, the state of the TT becomes "in translation", and then when the translator completes the translation, the state become "translated". At this point, the TT is assigned to a verifier, and its state becomes "in verification. Depending on the verifier's decision, the state of TT will then change to "rejected" or "completed". In the latter case, the creator is notified and then she or he can generate various formats of webpages; see Section 4.1.4. Fig. 2 shows a package in which the first two products are completed (they have not been submitted for translation because the creator provided her own translation), and Fig. 3 shows a package with the third task that has been translated (but not verified yet).

⊟ CD-P	pl_PL,en_US,	zh_CN,	2006-02-20 02:19	
	Product	**Type**	**Source-->Target**	**Last Modified At**	**Result of**
	product1	Document	pl_PL --> pl_PL	2006-03-16 23:09	User provided
	product2	Document	en_US --> en_US	2006-03-17 11:35	User provided
	Create Product				
⊞ package2	en_US,pl_PL,	zh_CN,en_US,	2006-02-28 14:32	
				[page 1 of 1]

Figure 2. Two completed products

Package names	**Source Languages**	**Target Languages**	**Last Modified At**	
⊟ CD-P	pl_PL,en_US,	zh_CN,	2006-02-20 02:19	
	TranslationTask	**Type**	**Source-->Target**	**Last Modified At**	**Status**
	task1	Document	pl_PL --> zh_CN	2006-03-19 15:56	translated
	Create TranslationTask				
⊞ package2	en_US,pl_PL,	zh_CN,en_US,	2006-02-28 14:32	
				[page 1 of 1]
Create Package				

Figure 3. A package with a translation task

4.1.3 Translators and Verifiers

Each translator and each verifier has a *profile***,** which is a list consisting of the source language, target language, and other specifications, such as the price of the translation and the translator's workload. We do not assume that the ability of translating from one language to another is reflexive. The preferred way to translate a text is to use a direct translation; however, if there is no translator who can perform this task, CGF

may choose to perform an *indirect translation*; first from the source language into an intermediate language, and then from this intermediate language into the target language.

When the translators logs in, she can use a GUI showing the list of submitted TTs, and then choose one of these tasks to start the translation process. The translator can work in one of two available modes; a batch mode and an interactive mode. The *batch mode* is useful for providing translations outside of the current framework of CGF, for example using specialized editors that support Chinese characters. In this mode, the translator downloads the source data, uses the separate tool to provide translations, and finally uploads the result to CGF. In the batch mode the translator can not reuse existing translations and she can create data that are not valid in the selected data definition (in the latter case, the translation would be rejected by CGF when it is uploaded). In the *interactive*, preferred mode, the translator works within the CGF framework, which displays a GUI, called a translation form, similar to the data definition form. Using this form, the source data can not be changed, and the translator can enter the translations, which are validated by the form for the selected data definition form. The translator can reuse existing translations or provide new translations. When and if these new translations are approved by the verifier, then they are incorporated into the repository of translations.

A translator can choose to start the translation process with *pre-translation*, which will use all existing translations from the translation memory for the source data in the selected TT (these translations can be later modified by the translator). The translator can also select the source phrase, and ask CGF to find existing translations of this word. Note that if the exact translation is not found, then *fuzzy* matching will be used (in the current implementation, all words starting with the same letter as the original word will be displayed). Fig. 4 shows the translator who selected a TT, opened a translation form, provided translations for the first book, and is currently translating data for the second book. In order to translate the word "Drama", the translator retrieved fuzzy translations of this word from the translation memory.

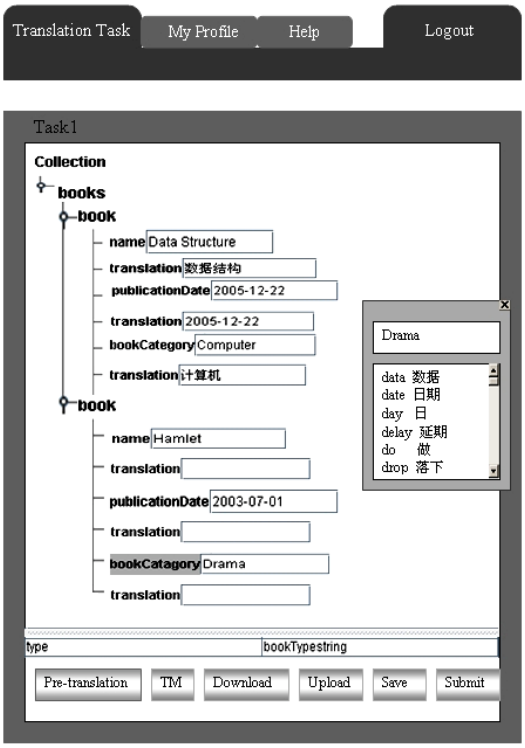

Figure 4. Using TM

The translator can save a TT, and when the translation is completed, she can submit it for verification (in the latter case, the translator can not modify this task any more).

The actions executed by the verifier are similar to these executed by the translator, except that initially she receives the translation task as created by the translator and can accept a translation (possibly with minor corrections), or reject it. Note that only *after* a translation is verified, the new terms are added to the

translation memory, and future translations can make use of these new terms. When the translation task state changes to “completed”, the creator can apply a rendering format; see Section 4.1.4.

4.1.4 Creating Multi-lingual Webpages

When a TT is completed, the creator can select it and apply a selected format for rendering. Available formats such as PDF or XHTML are stored in repositories. Fig. 5 shows a Chinese webpage in PDF..

Figure 5. Chinese webpage in PDF

4.1.5 Implementation of CGF

To implement CGF, we use XML (World Wide Web Consortium, 1994) because of its support for Unicode (Unicode Organization, 1991), separation of concerns (XML describes content rather than formatting), and ease of conversion between various formats. XML Schemas are used to implement data definitions. As a programming language for the implementation, we chose Java (Sun Microsystems, 2005), because of its support for internationalization (Deitsch and Czarnecki, 2001), and various built-in tools, such as servlets JSP (Sun Microsystems, 2005). Most importantly, we use Cocoon, and below we provide several examples of its applications.

The welcome page, from which the user selects a role, login name and password, is run by the main sitemap, which contains pipelines that implement the login and authentication. After login is completed, Cocoon uses different sitemaps for different kinds of roles. For example, if user logs in as a creator, Cocoon will use the sitemap for the creator. This sitemap contains a number of pipelines performing operations such as package management, and the creation of TT. The translation process uses the message catalog to implement translation memory, and the following two custom-built transformers: (1) a transformer to populate and update a message catalog, by reading two XML files respectively representing data in the source language, and the corresponding data in the target language; (2) a transformer to find available (possibly indirect) translators for the given source language and the given target language, by reading profiles of all translators and verifiers determining whether or not the translation can be performed, and if so what are the translations using the minimum number of indirect translations. Another example of a custom-built Cocoon component is a generator to validate forms, such as the form for creating a user account for which the information must be correct according to certain rules (among others, the user id must be unique). There are other forms, such as the form to create a package or a TT.

The persistent translation memory has been implemented using the native XML database, specifically Exist. Updates of the translation memory are performed through XUpdate (The XML:DB Initiative). Note that CGF can use any native XML or even XML enabled database as long as it supports XUpdate.

An essential part of the CGF is automatic generation of data definition forms from data definitions, a GUI-based modification of these forms, and the use of forms for creating translations. Therefore, the storage of data in XML and the use of XML Schema are transparent to the end-user. In addition, CGF checks the validity of data provided by the user.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we described the design and the implementation of the CGF, Cocoon-based system that can be used to globalize websites. Based on a data definition, the system generates a form that is used to enter only valid data, using a specific source language. Once the process of entering data has been completed, CGF takes care of submitting data to the translators and verifiers, and after the verification is done, it guides the user in the process of creating the required website.

Our future work includes developing various data definitions, standard taxonomies for data definitions for various domains, and tools to facilitate the creation of these taxonomies. Translation memory plays an important role in translation reusability and automatic pre-translation. We will develop standards for translation memory and, based on our experience, populate them with useful examples. In addition, we will investigate using the I18n Transformer and catalogs to internationalize the whole website, not only the document. For, example, for a creator whose default source language is Chinese, after the login, all the text on the page will be translated and displayed in Chinese.

In addition, this fall we will start the implementation of a different, P2P-based version of the globalization system (Müldner and Shen, 2006), in which there are two kinds of users. *Originators* advertise the work on selected documents and provide partial or complete contents in one ore more languages. *Contributors* can select advertised documents and provide more contents or translations to other languages.

REFERENCES

Aykin, N., 1999. Internationalization and Localization of the Web Sites. *Proceedings of HCI International on Human-Computer Interaction: Ergonomics and User Interfaces-Volume I. Munich, Germany*, pp. 1218-1222.

Belussi, A. and Posenato, R., 2004. *Internationalizing Data-Intensive Web Applications*. Rap. di ricerca RR 16/2004, Department Computer Science, University of Verona.

Benoit, D. and Müldner, T., 2004. IDUX: Internationalization of Data Using XML. *Proceedings of IADIS WWW/Internet 2004 Conference.* Madrid, Spain, pp. 469-476.

Deitsch, A. and Czarnecki, D., 2001. *JAVA Internationalization*. O'Reilly & Associates, USA.

Müldner, T. and Benoit, D., 2004. Generic Approach to Internationalization of Websites, *Proceedings of IASTED International Conference on Software Engineering and Applications SEA 2004.* Cambridge, USA, pp. 465-470.

Müldner, T. and Shen, Z., 2006. Cooperative Development of Internationalized Documents. COOP'06, 7th International Conference on the Design of Cooperative Systems. Carry-le-Rouet, France.

Müldner, T. and Shen, Z., 2005. Globalizing Internet Websites. The Ninth IASTED International Conference on INTERNET & MULTIMEDIA SYSTEMS & APPLICATIONS IMSA 2005~. Honolulu, USA.

Müldner, T., F. Wang and D. Benoit, 2004. My webpage can speak many languages. *AACE Proceedings of EDMEDIA'04*. Lugano, Switzerland, pp. 2040-2046.

Savourel, Y., 2001. *XML Internationalization and Localization*. Sams Publishing, Indianapolis, USA.

Sun Microsystems, 2005. *Java Technology*, [Online], Available: http://java.sun.com/ [Mar 2006].

Sun Microsystems, 2005. *JavaServer Pages*, [Online], Available: http://java.sun.com/products/jsp/ [Mar 2006].

The Apache Software Foundation, 1998. *The Apache Cocoon project*, [Online], Available: http://cocoon.apache.org/ [Mar 2006].

Unicode Organization, 1991. *Glossary of Unicode Terms*, [Online], Available: http://www.unicode.org [Mar 2006].

Webmail, [Online], Available: http://www.webmail.co.za [Mar 2006].

World Wide Web Consortium (W3C), 1994. *XSL Transformations (XSLT) Version 1.0*, [Online], Available: http://www.w3.org/TR/xslt [Mar 2006].

W3C, 1994. *Extensible Markup Language (XML)*, [Online], Available: http://www.w3.org/XML/ [Mar 2006].

The XML:DB Initiative. *XUpdate*, [Online], Available: http://xmldb-org.sourceforge.net/xupdate/ [Mar 2006].

Yu, Y. et al, 2003. Localizing XML Documents through XSLT. *Proceedings of the 21st IASTED International Conference on Applied Informatics*. Innsbruck, Austria, pp. 1059-1064.