Miha Filej; Michal Růžička; Martin Šárfy; Petr Sojka
Metadata Editing and Validation for a Digital Mathematics Library

In: Petr Sojka (ed.): Towards a Digital Mathematics Library. Paris, France, July 7-8th, 2010. Masaryk University Press, Brno, Czech Republic, 2010. pp. 57--62.

Persistent URL: http://dml.cz/dmlcz/702573

# Metadata Editing and Validation for a Digital Mathematics Library

Miha Filej[1], Michal Růžička[2], Martin Šárfy[3], and Petr Sojka[2]

[1] University of Ljubljana, Faculty of Computer and Information Science
Tržaška 25, 1000 Ljubljana, Slovenia
miha.filej@gmail.com
[2] Masaryk University, Faculty of Informatics
Botanická 68a, 602 00 Brno, Czech Republic,
mruzicka@mail.muni.cz, sojka@fi.muni.cz
[3] Masaryk University, Institute of Computer Science
Botanická 68a, 602 00 Brno, Czech Republic
sarfy@ics.muni.cz

**Abstract.** For preparing and validating metadata for the Digital Mathematics Library DML-CZ, a new tool, the Metadata Editor, has been developed. This paper outlines the procedures for Linguistic and geographical localizations its components. Also mentioned are such aspects as dynamic generation of form editing based on the XML Schema, the validation procedures as well as support for semiautomatic procedures regarding quality assurance.

**Key words:** DML-CZ, Metadata Editor, internationalization, translation, localization, validation, XML, forms, Ruby, Perl, JavaScript

## 1 Introduction

Since 2005, the Czech Digital Mathematics Library project (DML-CZ) [3] has been under development in the Czech Republic. An important part of the project has been the development of the Metadata Editor [4,11]—a client–server web application designed to manage, edit, and validate each article's metadata and full texts prior to their integration into the digital library.

The Metadata Editor is open-source software (`http://dme.sourceforge.net/`) and having proven its efficiency is now in use in a variety of other environments. These include the Faculty of Arts of Masaryk University, the Kramerius project of the Moravian Library [13], and the Editor may also be used in the EuDML project [5] as well. In this article we present some recent developments of the Metadata Editor.

## 2 On-line Submissions and Validation

The viability of a digital library rests with new acquisitions emerging mainly in the form of born-digital publications. The born-digital inputs to the Metadata Editor come from different sources, primarily from editors of various journals.

Petr Sojka (editor): DML 2010, Towards a Digital Mathematics Library, pp. 57–62.
 ISBN 978-80-210-5242-0

To assure a smooth integration of a new publication into the Metadata Editor Database, it has to satisfy a particular data format specification available to all the contributors. For this reason, it was necessary to set up a safe and comfortable interface between the contributors and the Metadata Editor. Because the Metadata Editor is a web application, it is easy to provide the users with direct on-line access based on a private user account in the Editor. After logging in the user can upload a new delivery directly and it is automatically assigned to the appropriate journal. The new entries are automatically validated so that the user gets warnings about inappropriate formats of data, while flawed submissions are completely rejected. It obviates later corrections and helps the users to prepare data in the required format.

## 3 Dynamic Generation of Editing Forms

One of the most important functions of the Metadata Editor consists in facilitating interactive modification of metadata. The operators are allowed to browse the contents of the Metadata Editor database and make necessary adjustments through the web-based interface of the relevant forms.

Since the metadata language is formally defined by an XML Schema, it is possible to generate the forms dynamically based on the XML Schema definition. The mechanism consists of server-side and client-side scripting. The XML Schema is processed on the server by a Perl script generating a JavaScript code that is included in the web page and which is subsequently sent to the client. This JavaScript code runs in the web browser of the end user and generates a form matching the language defined by the source XML Schema.

Not all features of the XML Schema are supported, but the mechanism is powerful enough to satisfy the requirements. In addition to being a part of the Metadata Editor, a generalized version of the forms generator is available as a standalone open-source project [9].

## 4 Internationalizing the Metadata Editor

### 4.1 Internationalization, Translation, Localization

In a nutshell, adapting the user interface of an existing application to new languages involves changing the output in a way that will please the current user. While translation could easily be considered the most important part of this process, it is not enough by itself—both translation and localization are required.

When dealing with source and target regions that are not similar, a complete localization of an application is difficult to achieve. Common parts of an application that have to be localized are time and date formats. The way time or date is displayed—the number of digits used, the separators, the order of date components, whether the 24- or 12-hour format is used—these can all vary from region to region. In addition, time zones in which users reside may

differ. The process of localization has to ensure that every date output of the application is displayed relative to the corresponding time zone.

Depending on the degree of internationalization that needs to be performed and the locales that need to be supported, more specific issues may be encountered: pluralization, units conversion (metric vs. imperial, currencies etc.), right-to-left text orientation. Particular attention has to paid to words or phrases that have different meanings due to cultural differences and may even be offensive.

### 4.2 Implementation

The Metadata Editor is built using a variety of technologies and programming languages. The part that interacts with the user is mostly handled by Ruby [8], which requires support from end libraries. In the past, there were various (incompatible) internationalization solutions in the Ruby ecosystem, each solving its own set of problems. In 2007 an effort to provide a generalized library emerged resulting in I18n [10], the library that is now the de facto standard for the internationalization of Ruby applications. Being a general solution it does not provide complex internationalization facilities; instead it defines an interface for other libraries to extend its functionality and remain compatible with each other at the same time.

I18n provides two basic methods, `I18n.translate` and `I18n.localize` (due to frequent use abbreviated to `I18n.t` and `I18n.l`, respectively). `I18n.t` handles translation by mapping an explicitly defined namespaced key to a string in a natural language. The approach differs from the popular GNU gettext [6] which maps a string in a natural language to a string in another natural language (although gettext's `.so` and `.po` files can still be used with I18n to store the translations). Having explicitly programmer-defined keys should result in greater maintainability by simplifying the way translations are reused throughout the application and avoids the issue where two sentences in different contexts in a language translate to the same sentence in another language, and vice-versa. `I18n.l` takes various objects like time, date etc. and localizes them according to the defined localization rules.

I18n's pluggable back-ends allow internationalized data to be stored in different ways. In addition to the gettext format mentioned above, YAML files, various relational databases and key-value stores are available as storage options. By defining the interface for implementing a back-end, the I18n library enables programmers to build a custom storage solution that suits their needs.

### 4.3 Choosing a Locale

Apart from altering the code to replace hardcoded strings with calls to methods that translate and localize them, a logic that handles switching between the locales needs to be introduced to the application. Since the Metadata Editor is a web application, the locale has to be set per request. With the help of sessions and cookies it is possible to persist a given locale between requests of the same

user, so the question remains: which locale is to be introduced for the first time (for a new user with no cookies)? There is no safe way how to determine a locale for a user's first request, but a web application is able to take a guess based on a few hints.

The HTTP/1.1 protocol defines the `Accept-Language` header [7] and at first it may be tempting to use the information provided by the user agent to set the default locale for the user, but there are several things to take into account [12]. Many users are unaware of the setting which was probably set when the user agent was installed and is might not conform to their preferences. The user agent may send a request that only defines the language without specifying the region (e.g. instead of `de-DE`, `de-CH` or `de-AT` indicating German as spoken in Germany, Switzerland or Austria, respectively, only `de` may be requested). If the user does not access the application from his own machine, the inferred locale may be inappropriate, especially when one is in a foreign country. Last but not least, the header may not be set at all.

Another clue from which the locale can be inferred is the user's IP address. With the help of a database or an external geolocation service it is possible to determine the user's geographical origin; but the approach shares a lot of the shortcomings described above. It is important that the application is not bound to its guess but allows the user to set his own preference at any point of the interaction. Whenever a locale is explicitly chosen, it is safe to assume it as a default for future requests from the same user.

To sum up, the logic for setting a locale has to consider (from highest to lowest priority): the previously set preference, the locale guessed from the HTTP headers, the locale guessed from the IP of the source and the default locale. Ideally the logic would set the locale as soon as the request was received, at the beginning of the interaction with the user, the programme input. Then, when computing the output, dedicated functions would perform translation and localization depending on the set locale.

### 4.4 Refactoring, Dangers, Precautions

The effort of adapting an application to another language rests with the difference between the source and the target language. Given that an adaptation to a broader set of languages is preferable, the codebase requires a major altering—a process that is prone to mistakes. The Metadata Editor being a relatively complex codebase, taking precautions against introducing bugs is the more important. The desired result of the process of internationalization is—(at least when rendering in the original locale) that the output matches the output of the programme before it was adapted. To assure this a set of specifications is needed.

Automated software testing is strongly encouraged in the Ruby community and the past few years have seen an evolution of tools and practices for unit, function and integration testing. In 2008 a tool called Cucumber [2] was introduced. It differs from other solutions in the way that specifications (called features) are not written in Ruby, but in a language called Gherkin [1]. This

domain specific language serves two purposes: documentation and automated tests. It allows describing software behaviour irrespective of how that behaviour is implemented. Gherkin's grammar has only a few simple rules and reads like spoken language. This allows feature specifications to be written and understood not only by programmers but by domain experts as well, thus increasing the value of the specifications. While Cucumber itself is written in Ruby, it can be used to test codes written in other languages, which makes it suitable to cover the non-Ruby parts of the Metadata Editor.

In Figure 1 one can see that Cucumber communicates with the application at the framework level, offering a better control over the request parameters than a direct communication with the application server or the web server level would provide.

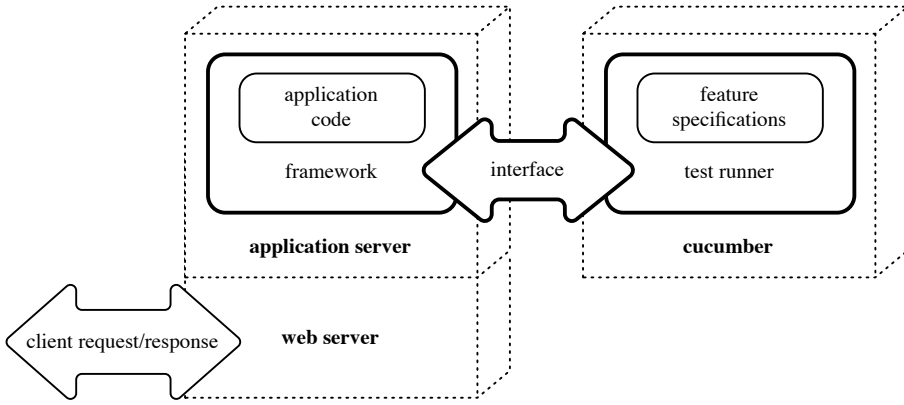


**Fig. 1.** Cucumber integration diagram

## 5 Conclusions

The Metadata Editor is a live, continuously developing project. New features are added as needed. The on-line input and validation service was worked in to provide users with a comfortable and safe interface for data inclusion, the user interface is dynamically generated based on the formal definition of the metadata, the localization of the Metadata Editor is in progress.

The Metadata Editor is used in several projects and will possibly be used in the EuDML project as well.

## References

1. aslakhellesoy / cucumber. [online], `http://wiki.github.com/aslakhellesoy/cucumber/gherkin`, Last edited by zwyan2009, 2 days ago [cit. 2010-04-28].

2. Cucumber : Behaviour driven development with elegance and joy. [online], http://cukes.info/, [cit. 2010-04-28].
3. Czech Digital Mathematics Library. [online], http://dml.cz/, [cit. 2010-04-24].
4. Digitization Metadata Editor. [online], http://dme.sourceforge.net/, [cit. 2010-04-28].
5. EuDML: The European Digital Mathematics Library. [online], http://www.eudml.eu/, This page was last modified on 20 January 2010, at 08:09. [cit. 2010-04-25].
6. gettext. [online], http://www.gnu.org/software/gettext/, Updated: $Date: 2010/01/31 14:51:43 $ [cit. 2010-04-28].
7. HTTP/1.1: Header Field Definitions : Accept-Language. [online], http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.4, [cit. 2010-04-28].
8. Ruby Programming Language. [online], http://www.ruby-lang.org/en/, [cit. 2010-04-28].
9. SchemaForms. [online], http://sforms.sourceforge.net/, [cit. 2010-05-30].
10. svenfuchs / i18n. [online], http://github.com/svenfuchs/i18n, [cit. 2010-04-28].
11. Bartošek, M., Kovář, P., Šárfy, M.: DML-CZ Metadata Editor : Content Creation System for Digital Libraries. In: Sojka, P. (ed.) DML 2008 – Towards Digital Mathematics Library. pp. 139–151 (2008), Birmingham, UK, July 27th, 2008.
12. Honomichl, L.: Accept-Language used for locale setting. [online], http://www.w3.org/International/questions/qa-accept-lang-locales, Last substantive update 2003-09-17 12:15 GMT. This version 2006-11-25 16:35 GMT [cit. 2010-04-28].
13. Šárfy, M.: Metadatový editor pro digitální knihovny. In: Knihovny současnosti 2009. pp. 140–154. Brno (2009), http://www.sdruk.cz/sec/2009/sbornik/2009-6-140.pdf, Seč u Chrudimi, CZ, June 23rd, 2009. ISBN 978-80-86249-54-4