Globalization Solution for a Web Based Community

A. Gut¹, I. Hoka¹, L. Miclea¹, G. Florescu², V. Florescu²
¹Technical University of Cluj-Napoca, 15 C. Daicoviciu, 400020 Cluj-Napoca, Romania
adrian.gut@aut.utcluj.ro, istvan.hoka@aut.utcluj.ro, liviu.miclea@aut.utcluj.ro
²National Institute for Research and Development in Informatics ICI Bucharest, 8-10 Maresal Averescu, Bucharest, Romania
gflores@ici.ro, valentin@ici.ro

Abstract—This article analyzes the technical, architectural and interaction challenges in implementing a globalization solution for a web based community. The purpose of this community is the integration of agrotourism resources and services in Romania and making them available to the mainstream.

I. INTRODUCTION

In a few short years the concept of “web community” has evolved rapidly. Various models have been tried, rules restated, concepts renamed and revised. Recently, a few models seem to have stabilized, and, as early adopters are moving on to newer things, web communities are becoming another useful tool for society.

While the ideas have settled now and the concept is ready to be consumed by the mainstream, some of the next challenges we face are to make it easy for people without a technical background to access and share information within these communities. One such challenge comes in the form of serving content and interacting with people in their native language.

Many web communities still rely on the fact that most of their targeted internet users are sufficiently proficient in the English language and the number of possible members they miss because of globalization issues is insignificant. Others accept the challenge of globalizing their content but, because of their inherent structure, they rely on a small percentage of members to contribute globalized content and these usually need a more advanced understanding of the underlying system. One such notable example is Wikipedia, which at the time of this writing consisted of some 5,300,000 articles in more than 100 languages. People from all over the world can contribute to Wikipedia by writing in a code that is simple to use but hard to grasp by those without some technical background. This is not an issue for Wikipedia however, because by the very nature of their content, contributors are usually professionals, part of academic or scientific communities.

The ARSIP project proposes to implement a portal that integrates agrotourism resources and services in Romania, promoting them on a national and international level. In order to do so, it is clear we must address the issue of globalization. Because of the way agrotourism works, we have chosen to model the portal in the form of a web community, where contributing members are those providing the services as well as those benefiting from them. We have introduced another type of member, one that serves the role of translator but does so without any advanced technical knowledge.

This article presents the solution we chose for globalizing the ARSIP portal, with the main purpose of delivering a system in which content is contributed and translated easily, without requiring technical knowledge beyond browsing the web.

A. What Is Globalization?

Globalization is the process of delivering software applications (in our case web based applications) in such a way that they can be used worldwide. Planning ahead for globalization while designing the application's architecture, improves the development process and saves work down the line [1].

Globalization involves two processes: internationalization and localization.

Internationalization is the process of designing the application so that it can be adapted to various languages without requiring any changes in the source code.

One of the most common issues with internationalization is deciding upon various character sets and encodings that are specific to each language. Proper internationalization requires that an application supports any and all possible character sets. The most convenient strategy for dealing with different character sets is to use UTF-8 encoding throughout the application, starting from the database and up to the user interface [2].

Localization is the process of adapting software for a specific culture by adding culture-specific components and translating text. Proper internationalization of an application is key to making it easy to localize for various cultures.

Implementing localization involves not only translating user interface elements but also formating information with respect to the selected culture; data that is subject to this formatting includes number format, decimal separator, number of digits after decimal, digit grouping symbol, displaying leading zeros, negative number format, currency, time format, AM/PM symbols, time separator, date format (short and long), date separator, measurement systems, a.s.o [3].

II. COMPONENTS AND ARCHITECTURE

A. Identifying Internationalization aspects?

Before deciding how to properly deal with various internationalization concerns, we must first identify and classify the data that needs to be localized. Three main types of data are revealed in our case [4]:

- Culturally-independent data. Such data is unchanged across various locales, even though their presentation changes. For example, fixed decimal numbers and timestamps (if standardized on UTC) are included in this class.
Sources for this type of data can be either the database or statically linked into the application code.

• **Static culturally-dependent data.** Data of this type can not change once the initial localization for a certain culture is done. Such data includes: application messages, user interface labels, online help and page layouts. This type of data exists in the presentation layer.

• **Dynamic culturally-dependent data.** This is the data associated with the entities in our domain model; most of the data volume in a data-driven application is of this type and most commonly stored in a relational database. Examples of this data include: products’ and services’ title and description, user comments and reviews, contributed articles a.s.o.

**B. Designing the Database to Support Any Number of Locales**

Many solutions exist for the problem of storing globalized data in relational databases. Some use special extensions of the underlying database management system (DBMS) to support globalization; others employ patterns for designing database schemas ranging from those that meet the purpose of any application requiring data representation in multiple languages to those that enable the translation of a few very specific properties. Choosing the right approach for design involves analyzing the tradeoffs between the various methods. Usually, the tradeoffs to be considered are flexibility vs. complexity vs. performance [5].

For the purpose of this project, we have chosen to develop our own pattern of structuring data, one that is modelled after the domain model of our problem space. Two main aspects have been considered in choosing this approach:

The domain model of our problem is relatively invariant over time, meaning that the structure of our entities will not change once the entity has been defined and, even more, there are no new entities added dynamically.

We are not aiming for supporting all known locales from the start, but we need to be able to add an arbitrary number of locales without overhead or penalty.

For these two requirements to be met, the database schema describing an entity must be sufficiently flexible to have some of its properties translated into any number of languages but can be rigid enough so that no new properties can be added to the entity.

**C. Pattern for Table Design**

The solution for storing our entities in a manner that allows us to have different representations of their properties in any number of cultures is to separate each entity into two distinct tables:

• one table would give the entity its identity and store all the properties that do not require different representation in other locales;

• the second table would hold all properties that have a different representation for every locale; the locale for which these properties are stored is also represented in this table.

These tables need to have a one-to-many relationship, where for each entry in the first table we can have any number of entries in the second table [6].

Here is an example of such a data representation, one that models a hierarchical structure of products and categories. The two entities here are *Category* and *Product*, having a one-to-many relationship: a product belongs to one category and a category can have any number of products. *Category* has only two properties: name and description – we want to be able to translate both of them; *Product* has four properties: name, description, price and code – we only want to translate name and description.

Following the pattern we described, our two entities are modelled (Figure 1) using four tables:

• *Categories* and *Products* are primary tables, holding primary keys and locale independent data. Only one row exists in each of these tables for a given product or category.

• *CategoryDetails* and *ProductDetails* are secondary tables, holding locale information and the properties which need to have a different representation for each locale. Foreign keys link these tables to the primary tables defining the entities. For each entity, there will be a row in this table for each locale in which data needs to be represented.

![Figure 1. Table structure of two entities with localized attributes](image-url)

Tables 1 through 4 show how two categories and two products would be stored given the table structure described.

**TABLE I Categories MAIN Table**

<table>
<thead>
<tr>
<th>CategoryID</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
</table>

**TABLE II CategoryDetails JOIN Table**

<table>
<thead>
<tr>
<th>CategoryID</th>
<th>Locale</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>en_US</td>
<td>Desktops</td>
<td>Desktop computers.</td>
</tr>
<tr>
<td>1</td>
<td>ro_RO</td>
<td>Calculatoare</td>
<td>Calculatoare de birou.</td>
</tr>
<tr>
<td>2</td>
<td>en_US</td>
<td>Laptops</td>
<td>Music albums, movies, TV shows, CDs and DVDs.</td>
</tr>
<tr>
<td>2</td>
<td>ro_RO</td>
<td>Laptpuri</td>
<td>Calculatoare portabile.</td>
</tr>
</tbody>
</table>
where data representation can be separated from data value:

Before designing the database schema, an important concern that must be addressed is to identify which values need to be stored in a culturally dependant form and which can be stored in a uniform representation that can later be converted by application logic. The process is shown in Figure 2.

Another, more flexible approach would be to extract text out of the web templates into resources and only translate the extractions. The process is shown in Figure 2.

### D. Data Storage vs. Data Representation

Before designing the database schema, an important concern that must be addressed is to identify which values need to be stored in a culturally dependant form and which can be stored in a uniform representation that can later be converted by application logic.

The following cases are common in data-driven applications, where data representation can be separated from data value:

- **Fixed decimal numbers** – English-speaking countries often format decimal numbers as 1,234.56, whereas people in many other countries format the same number as 1.234,56. Rather than maintain the original punctuation, a database attribute for such a value should be a coded decimal type that can later be presented in any format or encoding.

- **Strings** – The sequence of characters in a string, not the string's encoding, determines the string's value: For example, any number of different byte sequences can represent the string “abc”. Saving strings in a database in a variety of encodings, even if the encoding is stored with the value, can complicate processing the strings. The recommended approach for persisting strings received in multiple encodings is to use a universal encoding such as UTF-8 as the database attribute type, and convert from the received encoding to the database encoding before storing the value. The string can later be converted to other encodings for display. This way, issues like storing Cyrillic, Arabic or other non-ASCII characters are not a concern due to the UTF-8’s universal encoding capabilities.

- **Currency** – The importance of properly handling currency values cannot be stressed enough. The organization's business rules, not the user's locale, determine the values of quantities such as prices in a catalogue. If an application quotes a price in Yen to a Japanese customer, for example, the application should persist the value in Yen, not a value converted to U.S. dollars. (If business rules mandate conversion to dollars at the time of the quote, then the value should be displayed in dollars to avoid misleading the customer.) The application must always record currency values denominated in the currency mandated by business rules. When currency is converted, an audit trail often also requires storing the conversion rate and the value and denomination before conversion. An application's handling of currency values should always be checked by someone who understands the business's accounting rules. Extensive testing with audits can also uncover currency conversion errors.

- **Time and date** – Some global distributed applications standardize on a universal time coordinate (UTC) for all representations of dates and times, plus (optionally) an indication of time zone. Because UTC can be determined from local date and time for any geographic point, no data is lost in the conversion. As with currency, this determination depends on the organization's business rules.

Apart from those presented above, there are many more situations where data value and data representation may vary by locale. Uniform value representation in a database simplifies application coding, but should never cause information loss. Modern application frameworks contain a wide range of library calls that deal with proper formatting of data for the user, taking account of their locale [7].

By examining which properties of the entities in our domain model have the underlying data unchanged across locales, with only their representation being culturally dependent, we are able to decide how these properties are represented as table fields in our database schema. As such, culturally independent properties belong in the base table defining the entity, while culturally dependent fields belong in the join table.

### E. Globalizing Web-Pages

In our web-based community, we want to provide our readers with pages in their own native language. The simplistic approach would be to recreate the web templates in each language. However, this approach is not feasible in our case because it is labour intensive, would introduce unnecessary duplication into our code, making it difficult to maintain and evolve.

Another, more flexible approach would be to extract text out of the web templates into resources and only translate the extractions. The process is shown in Figure 2.

D. Data Storage vs. Data Representation

**TABLE III**

<table>
<thead>
<tr>
<th>ProductID</th>
<th>CategoryID</th>
<th>Price</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>899</td>
<td>DXPS410</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>699</td>
<td>E1505</td>
</tr>
</tbody>
</table>

**TABLE IV**

<table>
<thead>
<tr>
<th>ProductID</th>
<th>Locale</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>en_US</td>
<td>Dell XPS 410</td>
<td>Powerful entertainment with the latest technology standard</td>
</tr>
<tr>
<td>2</td>
<td>ro_RO</td>
<td>Dell XPS 410</td>
<td>Divertimento maxim la cel mai recent standard de tehnologie.</td>
</tr>
<tr>
<td>1</td>
<td>en_US</td>
<td>Inspiron 1501</td>
<td>Simple productivity and light use entertainment</td>
</tr>
<tr>
<td>2</td>
<td>ro_RO</td>
<td>Inspiron 1501</td>
<td>Productivitate ușoară și folosirea facilă pentru divertisment.</td>
</tr>
</tbody>
</table>
the page, like error and notification messages, static labels, general descriptions about the purpose of the or how the user should interact with them. These are values that must be displayed differently for every user depending on their native language. The underlying data is essentially the same; it just has a different representation depending on locale. Such data is called globalization resources and can be extracted from the rest of the page [8].

Once we have extracted all the globalization resources and stored them in a resource file, it can be forwarded to an authorized translator.

Identifying static text in the original page allows us to replace it with placeholders resulting in a web template that uses resources. The translation process (step 3) will result in resource files, for every supported locale. At processing time (step 4), the web template will extract the required resources depending on the selected locale and output a fully translated web page [9].

Another issue we must consider is how dates and times, numbers and currency are formatted. Fortunately, this is handled seamlessly in most of web development platforms, by using built-in formatter classes, when a culture or locale property is set.

F. Integrating the Solution

Having presented the components necessary for building a globalized application that meets our requirements, we can now determine where they fit in a 3-tiered data driven application (Figure 3) [10].

**Data layer** - holds the globalization data, identified as being dynamic and modelled after the pattern we described.

**Application Logic** – is responsible for retrieving and handling globalization dynamic data, according to the current locale setting [11].

**Presentation layer** - data formatting and static translations are performed as required. All users in the system interact with this layer, regardless of their role.

![Figure 3. Using 3-tiered architecture to implement the globalization solution](image)

### III. INTERACTION FLOW

One of the main goals of our system is to mediate the interaction between two primary types of content contributors: accommodation owners and translators.

In order to make the interaction as seamless as possible, the system must enforce a set of rules based on its knowledge about roles, making decisions based on minimal user input. At any given point, the next steps available to users are clearly defined and the user is exposed only to the options that are available [12].

Accommodation owners define the source content in their native language and request for it to be made available in other languages as well. Source content includes: accommodation name, description, available services, facilities, room configuration, availability and pricing.

Translators contribute localized content based on the content defined by owners and can review content contributed by other translators. **Figure 4** describes how accommodation owners and translators interact within the localization process.

**A. Accommodation Owner Interaction**

Accommodation owners start the localization process by defining content in their native language and requesting for their content to be translated in other languages. These two processes are exemplified in **Figure 5**.

Once a translation has been requested, owners are able to monitor their state of completion. A translation can be in any one of the following states:

- **Not assigned** – means the translation has been requested, but no translators have accepted the task yet.
- **Assigned** – a translator has accepted the task but is not working on it yet (no drafts have been saved).
- **In progress** – translation has started.
Submitted – the translator has finished the task and submitted the work for review.

When the translation is in the submitted state, the owner has two options: approve and publish the translated content, or reject the translation and request another translator to accept the task (Figure 6).

B. Translator Interaction

Translators can access translation tasks requested by owners in the languages that match their profile. They are able to accept translation tasks (Figure 7), perform partial translations or finalize and submit (Figure 8).

For translations submitted by other translators that match their language profile, translators may perform reviews, informing the accommodation owner about the quality of the work (Figure 7). Also, they are able to monitor the status of their submitted translations through the activity center.

C. Published Results

Translating content reaches the public area only after they have been reviewed by peers and approved by accommodation owners. As more translations become available, links appear on the site for each localized version (Figure 9).

IV. CONCLUSIONS

The solution presented in this article addresses the issue of globalization for a web-based community where members need not have a technical background, in an effort to build a community that is accessible to the mainstream. Here are the main points that make such a system possible:

• Flexible data representation, supporting dynamic addition of locales;
• Clear workflow for contributing new content and localizing it;
• Peer-based review system to ensure high quality for localized content;
• Automatic, behind-the-scenes management of localization rights, only exposing localizable content to users that fit the profile [13].

First, we analyze the technical aspects of internationalization, deciding on the best data representation for our content and how to architecture the application logic so that localization is possible without significant effort.

In the second part we define a way in which members of the community should interact with our system, focusing on the steps that involve localization.

ACKNOWLEDGEMENTS

This work has been supported by the Romanian Education and Research Ministry under the contract no. 17/2005 "Agro-tourism Resources and Services Integration Portal - ARSIP"

REFERENCES


