Internationalizing Data-Intensive Web Applications

Alberto Belussi
alberto.belussi@univr.it

Roberto Posenato
roberto.posenato@univr.it

Questo rapporto è disponibile su Web all’indirizzo:
This report is available on the web at the address:
http://www.di.univr.it/report
Abstract

Nowadays a considerable amount of web sites are data-intensive web sites, meaning that web pages are generated by extracting their content on the fly from a database system. Systems of this type are often the result of an integration process between intranet legacy systems and Internet web applications that represent the front end of the organization towards the rest of the world.

The globalization process, that influences the activities of any organization (company, public agencies, etc.), makes it necessary to provide information in different languages in order to enlarge the audience of the information presented through the web. Therefore, it is often required to extend the web system for handling information in more than one language. This process is called internationalization of an application or system. Another important requirement in this process is that it should be applied without a strong reorganization of the web applications that implements the web site, so that the main cost would regard the translation of the web contents.

We propose a general framework for the internationalization of data-intensive web application based on the MVC-2 paradigm and a relational database system. Moreover it can be applied to existing web applications with a low impact on code restructuring. This framework has been applied at the University of Verona (Italy) for the internationalization of the web sites of all departments and faculties of this university.
1 Introduction

In the last few years the globalization process has influenced the activities of any kind of organization (companies, public agencies, universities, etc.) around the world. One of this process effects concerns the production of any kind of information in more than one language in order to enlarge as much as possible the audience on the world market. This observation is true in particular for web sites contents and it often leads to an extension of the existing web applications in order to enable them to handle information in more than one language. However, this extension could be performed by following several very different approaches and with results of different quality. In general we call the adaption of an application to support a multilingual environment \textit{internationalization of an application}. The process of adaption of an application for a specific region or language by adding local-specific components and translating text is called \textit{localization of an application}.

The internationalization of a web application can be a hard task, in particular if the application architecture does not conform to a standard schema, like the Model View Control (MVC) approach, or does not separate at least the logic and the presentation in different modules. Considering web applications with a well-formed architecture, the internationalization should not require a strong reorganization of the software, but the main cost should regard only the translation of the web site contents.

In this paper we propose a general framework for the internationalization of data-intensive web applications [8, 7] based on the MVC-2 paradigm [4] and one or more relational database systems. This framework can also be applied to existing web applications with a low impact on code restructuring.

Previous works on data-intensive web applications regards mainly the definition of a design methodology and the implementation of case tools [9, 7, 6, 8]. Many other papers in the human-computer interaction area face the general problem of internationalization and localization of data presentation in web sites, considering in particular aspects like symbols or color perception in different cultures, format conventions (for example, date or time presentation), etc. [5].

To our knowledge, the presented framework is one of the first proposal regarding a systematic approach for the internationalization of data-intensive web applications. In particular it focus on the impact that the internationalization has on database schema, queries specification and query results handling.

The rest of the paper is organized as follows. In Section 2 the problem of internationalization of a web application is described. In Section 3 the proposed framework is presented by describing in details its components: a database extension method, a query rewriting technique, a method for managing state objects with translations in different languages, a mechanism to handle language in HTTP request and some criteria to generate partially translated pages. In Section 4 we briefly illustrate the result obtained by applying the framework to a real case: the web sites of the faculties and departments of the University of Verona (Italy).
2 Problem definition

In this paper with the term “web application” we refer to any software application that is accessible using a web browser. Its main characteristic is that it produces web pages (i.e. HTML files) that are sent to the browser through the net under the HTTP protocol. During the last few years the architectures of web applications have evolved dramatically and on the net we can find several types of web applications: some of them produce pages from static HTML files, others generate pages extracting data from databases and work in a complex multi-tiered environment [8].

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). This problem can have different solutions with respect to:

1. the architecture of the web application, we distinguish between the following two cases: applications based only on static HTML files, that need to be updated manually by human experts (we call them Static Web Applications); applications based on a template system, that is able to produce web pages dynamically on request, which means that the page content is extracted from one or more databases and the HTML code, constituting the page, is produced on the fly and sent to the browser (we call them Data-Intensive Web Applications). Figure 1 shows the architecture of this type of application.

2. the level of update between the master pages and the translated ones, that we intend to maintain: the target may be to accept a partial discrepancy between the master page and its translations or to keep them always in a consistent state.

According to the architecture, different solutions of the internationalization problem can be applied, in particular:

1. Static Web Applications: in this case the replication of the static HTML files with the translation of their content in the languages of interest is the widely used solution; of course, the maintenance of the consistency between the different replications can become a time consuming task; therefore, in some cases, the organization can decide to accept some discrepancy between the pages in the master language and the translated pages. This usually leads to parallel web sites where the site versions in foreign languages contain only a part of the information presented in the site in the master language.

Notice that the replication can be implemented in different ways:

- by performing a copy of the whole application directory tree for each foreign language we want to handle;
- by adopting a simple mechanism of “server driven content negotiation” (according to the protocol HTTP/1.1), which allows one to keep in one directory tree all the HTML files for the different languages, with a suffix that identifies the language.
by adopting a more sophisticated mechanism of “server driven content negotiation”, that allows one to represent in one file all the different translations of a file content [1].

2. Data-Intensive Web Applications: in this case the internationalization can be obtained by following different approaches. The replication is still a chance and it requires to generate one or more copies of the whole system. This means that is necessary to replicate the database and the portion of the code that produces the web pages (in particular script files containing static text like JSP, ASP, PHP, etc.), since both of them are subject to translation. The implementation of the replication approach can follow the three solution presented above. Replications in this case are more complex to maintain since it is necessary to keep consistency among more independent database instances.

An alternative solution is to make the application language independent. This can be achieved by a revision of the whole application and, in particular, by applying the following three steps:

- **Database extension**: the database has to be extended in order to store the translations of data that are subject to change with respect to the language,
- **Queries rewriting**: the queries have to be rewritten to take into account the new tables containing data translations.
- **Static text translation**: the static text inside script files (like JSP, ASP, PHP, etc.) has to be replaced by a call to a procedure that computes the right translation according to the required language.

![Figure 1: The reference architecture for data intensive web applications.](image)

We propose a framework for making a data-intensive web application language independent. In particular, the framework can be applied to applications adopting the MVC-2 paradigm [4] as described in the following section.
3 The proposed solution

The internationalization of a data-intensive web application based on the MVC-2 paradigm can be performed by exploiting the architecture schema that the application inherits from this paradigm.

In particular, we suppose to find in the application the following components:

- **the Controller**: it receives an HTTP request and decides which actions have to be activated to serve the request. Actions are contained in the model component and they are generally object oriented components themselves.

- **the Model**: it is composed of a set of object oriented components (for instance a set of Java classes), implementing the actions to be performed for handling the requests. In particular, we expect to find a set of components that submit queries to the database server and handles the results, and a set of components, called *state objects*, that store the query results (for instance a set of Java Data Beans).

- **the View**: it is composed of a set of components, representing script files (JSP, ASP, PHP, etc.) that are responsible to build the HTML file containing data coming from state objects created by actions of the model.

The proposed framework has been studied in order to obtain the following results:

- it should make the application language independent in the sense that the application should work with a certain set of foreign languages and this set can be extended without requiring any code modification.

- it should allow one to extend the set of attributes subject to translations without requiring any code modification.

- it should require the minimal database extension, but, at the same time, the minimal effort in query rewriting and the minimal performance worsening in query processing.

Hereby we describe in details the proposed frameworks considering: database extension and queries rewriting, internationalization of state objects, mechanisms to handle language in HTTP request and partial translation management.

3.1 Database extension and queries rewriting

The main characteristics of a data-intensive web application is the interaction with one or more database servers, which provide dynamically the data to be published in the requested web pages. This approach always guarantees web pages with up to date content. The internationalization process requires the translation of the information store in the database, thus an extension of the database schema is necessary in order to store such translations.

We suggest the following general approach to extend the database schema, by supposing to adopt the Entity-Relationship model for the conceptual representation of the database schema.
1. For each entity type \( E \) of the schema the attributes subject to translation in at least one foreign language have to be identified.

2. For each entity type \( E \) with attributes \( \{A_1, \ldots, A_n\} \) having some attributes \( \{A_{i_1}, \ldots, A_{i_m}\} \) subject to translation, we specify a new entity type \( E_{\text{trans}} \) weak entity of \( E \), with the following attributes \( \{\text{language}, A_{i_1}, \ldots, A_{i_m}\} \) (Figure 2).

3. For each relationship type \( R \) of the schema the attributes subject to translation in at least one foreign language have to be identified.

4. For each relationship type \( R \) among entity types \( \{E_1, \ldots, E_n\} \) with attributes \( \{A_1, \ldots, A_n\} \) having some attributes \( \{A_{i_1}, \ldots, A_{i_m}\} \) subject to translation, we replace \( R \) with a new entity type \( E.R_{\text{trans}} \), weak entity of \( E.R \), with the following attributes \( \{\text{language}, A_{i_1}, \ldots, A_{i_m}\} \) (Figure 3).

Considering the implementation in a relational database system, which is the reference implementation system of the proposed framework, the schema extension above described has the following impact:

- for each table \( E \), representing an entity type with attributes subject to translation, a new table \( E_{\text{trans}}(K_E, \text{language}, A_{i_1}, \ldots, A_{i_m}) \) has to be created (\( K_E \) is the set of attributes representing the primary key of \( E \)).

- for each table \( R \), representing a relationship type with attributes subject to translation, a new table \( R_{\text{trans}}(K_R, \text{language}, A_{i_1}, \ldots, A_{i_m}) \) has to be created (\( K_R \) is the set of attributes representing the primary key of \( R \)).

Thus, only an additional table for each entity or relationship type subject to translation has to be added.

![Figure 2: Database extension for entity types translation.](image)

Thus, only an additional table for each entity or relationship type subject to translation has to be added.

Regarding database schema extension other solutions could possibly be considered. However, we will discuss them after the following presentation of the query rewriting technique, since they have an impact also on that task.
According to the proposed framework, a query rewriting is necessary in order to include the translated attributes in the result of each query. The characteristics of this rewriting technique are the following ones:

- It preserves the result cardinality that the query had before the internationalization.
- It uses only two languages $L_1$ and $L_2$ as query parameters. These two languages are obtained by applying a custom rule to the preference languages list coming from the HTTP request (see Section 3.3). We denote $L_0$ as the master language, $L_1$ as the requested language and $L_2$ as the alternative language. We could consider also more than two languages as query parameters, but it is very unusual that an application has to produce a page containing information in more than three languages.
- It extends each query result schema by applying the following rule. For each attribute $A$ subject to translation, we replace $A$ with three attributes $L_0$-$A$, $L_1$-$A$ and $L_2$-$A$, containing its values in $L_0$ and its translations in $L_1$ and $L_2$ respectively.

Given a generic query:

\[
\text{SELECT } E_1.A_1, \ldots, E_k.A_n \\
\text{FROM } E_1, \ldots, E_m \\
\text{WHERE } C
\]

suppose that the table $E_1$ has a translation table $E_{1\text{trans}}(K, \text{language}, A_1)$, then the above query must be rewritten as follows:

\[
\text{SELECT } E_1.A_1 \text{ AS } "L0-A_1", L1.A_1 \text{ AS } "L1-A_1", L2.A_1 \text{ AS } "L2-A_1", \ldots, E_k.A_n \\
\text{FROM } E_1 \text{ LEFT JOIN } E_{1\text{trans}} \text{ AS } L1 \text{ ON } E_1.K = L1.K \\
\text{LEFT JOIN } E_{1\text{trans}} \text{ AS } L2 \text{ ON } E_1.K = L2.K, \ldots, E_m \\
\text{WHERE } C \text{ AND } L1.\text{language} = ? \text{ AND } L2.\text{language} = ?
\]

Notice that, after the rewriting process, the resulting query contains some parameters, represented as "?", which stand for the requested language $L_1$ and the alternative language $L_2$ respectively.
This rewriting technique can be trivially extended to the general case in which table $E_1\text{trans}$ contains more than one attribute and there are other tables $E_i\text{trans}$. Moreover, given a query containing $m$ tables with attributes subject to translation, the number of outer joins that have to be added to the query is equal to $2m$.

We discuss now some alternative solutions to solve the problem of storing the attributes translations in the database. Indeed, the above described extension mechanism represents a middle point between two limit approaches:

1. **Single translation table approach**: the translations are stored in one additional table $\text{Translations}(\text{table, attribute, language, } K, \text{value})$. This solution can be applied only if attributes $K$ can contain the primary key of every table of the database (this can be done only if every table has the same set of attributes as unique identifier, for instance an attribute called $ID$). The main drawback of this solution is that the query rewriting requires the introduction of a high number of outer joins. In particular, given a query containing in the select clause $n$ attributes subject to translation, then the number of required outer joins is $2n$. This will considerably reduce the performance of query processing, since outer joins are very expensive operations and the cardinality of the $\text{Translations}$ table could be very high, given that it contains all database translations.

2. **Additional translated attributes approach**: the translations of an attribute are stored in additional attributes (one for each language) added to the original table. No additional tables are required. In this case the database structure is language dependent in the sense that, if we want to add the translation of an attribute in a new language, we have to alter the database schema by introducing attributes to store the new translations. This may cause the generation of many null values inside the additional attributes. Moreover, the query rewriting requires to extend the list of attributes in the select clause including all the translation attributes without any language selection and, again, adding translation in a new language, requires query rewriting. Notice, that the advantage of this approach is the fact that queries do not use outer joins.

3.2 Internationalization of state objects

In a MVC-2 web application, state objects represent both the request from the browser and the data needed to build the response. In a data-intensive web application, state objects are used also to represent tuples of queries results. In this section we consider how to internationalize this latter kind of state objects.

The internationalization process requires to extend the state objects in order to represent the translated values of object properties subject to translation. Given the goal of applying language independent extensions, the more suitable way to do the task is to insert, for each property $A$ subject to translation, a new property $A$-lang that will contain the language name (or language code) of the value stored in $A$. In this way a state object can store values in any language with a minimal impact on its type restructuring. The trivial approach that adds a new type of state object for each language is obviously language dependent, thus we do not consider it as a possible solution.
Given a tuple \( t \) of a query result \( Q \), having translated values stored in the same tuple, a state object \( O \) is created and is populated by applying the following approach:

- for each attribute \( A \) of \( Q \) subject to translation choose one translated value in \( t \) according to a custom rule;
- put the chosen value in the corresponding property \( A_i \) of \( O \) and its language-name into property \( A_i\text{-lang} \);
- finally, put in the corresponding properties of \( O \) the other values belonging to attributes not subject to translation.

Therefore, each state objects constructor having a query result tuple as parameter, have to be modified according to the new proposed approach. Nevertheless, this process can be avoided if all constructors based on query result tuple are replaced with a single meta-method \( \text{populate} \). Method \( \text{populate} \) works as follows. Given a state object, a query result tuple, the requested language \( L_1 \) and one or more alternative languages, it fills the state object with tuple values according to \( L_1 \) and the alternative languages. \( \text{populate} \) can be implemented under the following conditions:

1. The query result is written as described in Section 3.1.
2. The state object representing a tuple of a query result \( Q \) contains
   - (a) at least as many properties as the attributes of \( Q \) where each property has the same name of the corresponding attribute of \( Q \);
   - (b) a property for the language-name \( A_i\text{-lang} \) for each property \( A_i \) subject to translation.
3. The adopted programming language provides introspection mechanism for the state object and the query result object; the introspection mechanism permits: to discover properties and methods of an object at run time and to access them dynamically (i.e. in order to execute a method first you retrieve its signature by a meta-method and then you invoke it by another meta-method). For example, Java language Reflection/Introspection mechanism permits such operations.

The pseudo-code of \( \text{populate} \) method is shown in Algorithm 1. In the algorithm we limit the alternative languages to one \( (L_2) \), we omit the handling of exceptions for sake of readability and we hide details of the introspection operations by method calls. Notice that:

1. If there is no corresponding property in the state object for a query result attribute, the method \( \text{copyProperty()} \) should throw an exception because it is not known where to store the attribute value.
2. After the execution of \( \text{populate} \), some properties can be uninitialized since the algorithm fills only the ones corresponding to query result attributes. Therefore, to improve the soundness of the state objects, it is necessary to add them a control that throws an exception when a property without a value is accessed.
Algorithm 1 populate

/*
 * 'bean' is the state object to fill,
 * 'rs' is the query result represented as a array of tuple
 * 'lang' is an object that represent \( L_1 \) and \( L_2 \)
 * in this algorithm, column is synonym of attribute of query results
 */

INPUT bean, rs, lang
//Acquire result set meta data
metaData = rs.getMetaData();
#cols = metaData.getColumnCount();
if (rs != null) then
   for all (1 \leq i \leq #cols) do
      columnName = metaData.getColumnName(i);
      columnType = metaData.getColumnType(i);
      if (lang != null && columnName.startsWith("L0-")) then
         //Put into translation[] the next 3 values of the query columns
         columnName = columnName.substring(3);
         translation[0] = resultSet.getString(i++);
         if (!metaData.getColumnName(i).equals("L1-"+columnName)) then
            throw exception! //there is not the first translation column
         end if
         translation[1] = resultSet.getString(i++);
         if (!metaData.getColumnName(i).equals("L2-"+columnName)) then
            throw exception! //there is not the second translation column
         end if
         translation[2] = resultSet.getString(i++);
         //Choose the right value according to a custom rule using
         //lang parameter
         columnValue = getPreferedValue(translation, lang);
         languageChoosen = getLanguageChoosen(translation, lang);
         //Put the columnValue in attribute with name columnName
         //by copyProperty method that exploits introspection mechanism
         copyProperty(bean, columnName, columnValue);
         copyProperty(bean, columnName+"-lang", languageChoosen);
      else
         //normal attribute
         columnValue = resultSet.getObject(i);
         copyProperty(bean, columnName, columnValue);
      end if
   end for
end if
OUTPUT bean
3.3 Handling language in HTTP requests

The language to use for building the response to a HTTP request should be chosen as to the user preference.

The HTTP/1.1 protocol specifies that a HTTP request can have a header field, the \texttt{Accept-Language} field, that represents the weighted list of the languages that are preferred as a response to the request. A user can set this header by setting the browser languages option. We call this language(s) preference setting \textit{implicit language choice}.

Since the \texttt{Accept-Language} is an optional header field, not all browser have the languages option. To get round this problem and to allow a user to set the preferred languages, in the web site there should be always a section where the user can choose the preferred language(s). We call such a choice \textit{explicit language choice}. An explicit language choice should persist for all the session length.

By the implicit language choice, you can receive a list of languages for which there are no translations. On the other side, by explicit language choice, it is more user-friendly to ask the user to select only one language even if it is useful for the application to have also an alternative one. Therefore it is necessary to implement a logic rule that manages these cases and builds a state object \textit{Lang} that represents at least two of the user preferred languages consistently with the available translations. \textit{Lang} will be used by other logic modules of application to choose the right translations.

3.4 Criteria for generating partially translated pages

The proposed query rewriting technique has the aim to extract from the database the requested data together with their translation in two languages $L_1$ and $L_2$. Considering that $L_1$ represents the requested language, the application should be able to face different cases:

- all attributes subject to translation are available in $L_1$.
- a subset of attributes subject to translation are available in $L_1$ and the rest in $L_2$.
- a subset of attributes subject to translation are available in $L_1$, a subset in $L_2$ and a subset has no translation.

The application can apply different strategies to handle the second and the third case. The strategy of minimum impact simply shows the page in two languages for the second case. For the third case, a page in three languages will be produced showing the attributes with no translation in the master language (“babel approach”).

However, there can be alternative solutions that, for example, avoid to show the non translated attributes, obtaining at most two languages in the same page, or that show pages only in one language $L_1$ or $L_2$. This latter solution can be applied, for example, by choosing $L_1$ (and alternatively $L_2$) only if all mandatory attributes are available in $L_1$ ($L_2$).
4 The internationalization of the web site of the University of Verona

The proposed approach has been applied for the internationalization of the web sites of the faculties and departments of the University of Verona (UNIVR). The UNIVR web system is organized as follows:

- A central site (http://www.univr.it): it is composed of a set of static HTML files.

- A set of 8 faculties sites (http://www.scienze.univr.it for the Science Faculty site): they are generated by a data-intensive web application (faculty on line - FOL).

- A set of 24 departments sites (http://www.di.univr.it for the Computer Science Department site): they are generated by a data-intensive web application (department on line - DOL).

- Both FOL and DOL are developed according to the MVC-2 paradigm as a set of Servlets [2], Java classes and JavaServer Pages (JSP) [3]; moreover, they extract data from one unique database containing the information about the teaching and research activities of the whole University.

- A web application for updating the database content with restricted access that permits to distribute the effort of updating information among all the offices and persons working at the University (teachers, secretaries, PhD. students, researchers, etc.)

We have applied the proposed framework for the internationalization of FOL and DOL during 2003 and now the faculties and departments sites can show pages in four foreign languages (English, French, German and Spanish) plus Italian (the italian and english versions of the home page of the science faculty are shown in Figure 4).

The impact of the internationalization process of the UNIVR web system can be described as follows:

- the database schema, that contained 275 tables, now contains 340 tables.

- the Java class of FOL (DOL), that is responsible to interact with the database, contains 117 SQL queries (the same number of queries were present before internationalization) and it contained about 7000 and now about 7400 code lines.

- there were 43 methods in FOL (DOL) for Java Bean object population, now there is only one method Populate in each application.

- the static text contained in the JSPs of FOL and DOL has been internationalized by adopting the Java internationalization mechanism based on bundle files.
Figure 4: The home page of the Science Faculty site in Italian and English.
5 Conclusions

In this paper we have illustrated a framework for the internationalization of a data-intensive web application. The proposed solution has been studied for an application based on the Model-View-Controller 2 paradigm and interacting with a relational database system. The framework was successfully applied for the internationalization of the web sites of University of Verona. This implementation is composed of two applications based on Java Servlet and JavaServer Pages technology, but it is very general and can be used also in other template systems like .NET technology.

References
