Towards an Evaluation Framework for Multilingual Supported Data Modeling Patterns

Gholamali Nejad Hajali Irani
Computer Engineering Dep.,
University of Bonab
Velayat Avenue, Bonab 5551761167,
East Azerbaijan, Iran

Mohammadreza Rostamzadeh
Lachin Systems Company
Rajayi Avenue, Bonab 5551867311,
East Azerbaijan, Iran

ABSTRACT

Nowadays, Internationalization is one of the most important aspects of Information Systems. Supporting multilingual features as a part of internationalization process, is being a competition in all Information Systems. There are many data modeling patterns to support multilingual features in development process of Information Systems with many quality attributes to evaluate them.

In this article, an evaluation framework has been designed for all multilingual support data modeling patterns and their quality attributes.

To obtain this aim, all data modeling patterns to support multilingual features have been collected. Then, all quality attributes related to patterns are collected and categorized as evaluation parameters. Afterwards, all quality attributes for each pattern been investigated. Finally, an evaluation framework has been tried to provide for all multilingual patterns and their quality attributes.

General Terms

Software Engineering, Data Modeling Patterns.

Keywords

Internationalization, Multilingual Information Systems, Data Modeling, Quality Attributes.

1. INTRODUCTION

Internationalization (i18n) is the tasks and activities to provide services so that systems can easily be adapted to any local languages and cultures, that activities called localization [1]. In computing, internationalization and localization are means of adapting computer software to different languages, regional differences and technical requirements of a target market (locale). [2].

Multilingual is part of internationalization process. Information Systems (IS) should support multilingual feature in their Information Systems in order to be included in world-wide competition.

Multilingual is explained divided into two categories. Firstly, Software or IS should have a multilingual User Interface (UI) and its UI should be able to seem multilingual. Secondly, IS should store the data of IS in different languages. Therefore, multilingual feature affect all parts of IS development process. The second category is the scope of this article.

Data Modeling is one of the most important steps of Software Development Life Cycle (SDLC) [3]. In this step, the structure of data for IS will be analyzed and designed by software analysts and designers. Data modeling patterns are common data modeling structures that occur in many data models [4]. To support multilingual features in any IS, the data model of IS should support multilingual features.

There are many data modeling patterns to support multilingual features in ISs. Architectures and designers of ISs should be aware of those features to design a multilingual IS. Also, they should know the weaknesses and strengths of each pattern. In order to measure these weaknesses and strengths, architectures should evaluate the patterns. Evaluation parameters of patterns are called quality attributes [5]. Therefore, multilingual patterns are measured based on their quality attributes.

In this article, the beginning steps of providing an evaluation framework to compare and evaluate all multilingual data modeling patterns have been presented as follows:

1. Gathering and categorizing all quality attributes as evaluation parameters related to multilingual support data modeling patterns and also their investigation and quantification.
2. Gathering all multilingual data modeling patterns and investigating and measuring the value of each quality attribute for each pattern.
3. Providing an evaluation framework for all patterns.

2. EVALUATION PARAMETERS

In software engineering, there are many quality attributes to compare and evaluate the strengths and weaknesses of patterns can be named as evaluation parameters. In this section, some important quality attributes of multilingual support data modeling patterns are investigated. According to the concept of data modeling, these quality attributes are categorized in two major sections. First category of quality attributes are related to Software Architecture named SQ. The second one quality attributes related to Database Management Systems named DQ. In the following sections, each quality attribute is described.

2.1 SQ1: Extendibility for adding new languages to the system at development-time

This quality attribute is related to the complexity of adding new language to the system at development-time. The complexity of adding this language can be a number between 0 and 100.

2.2 SQ2: Extendibility for adding new language to entire system at runtime

Same as previous, this quality is related to the complexity of adding new language to the system at running time. The complexity of this action can be a number between 0 and 100.

If a pattern does not support this quality attribute, the number is set to 0. In the best case, a new language can be automatically added in runtime, which set the number to 100.

2.3 SQ3: Modifiability for deleting an existing language from entire system at develop time

This quality attribute is related to the complexity of deleting an existing language from the entire system at development time. Again, it can be a number between 0 and 100. This action should rollback the state of system to a stable state and all performance issues should be eliminated from the system.

2.4 SQ4: Extendibility for adding new language to entire system at runtime

This feature is the same as previous one, but all requirements should be executable at running time.

2.5 SQ5, SQ6, SQ7 and SQ8

All these quality attributes are similar to SQ1 to SQ4, but they are related to specific field of a table; again, their values can be a number between 0 and 100.

2.6 SQ9: Usability for just default language

This quality attribute is related to the usability of a default language in a multilingual circumstance. Other words, by installing and using multilingual systems, the structure of source codes and other programming issues for using only the default language should not be affected. In the best case, adding the multilingual patterns do not affect any code in the system for the default language and the value of pattern would be 100. The ratio of changing the code in comparison with changing other patterns, determine the value of this pattern.

2.7 SQ10: Modularity of multilingual pattern

The modularity of adding multilingual patterns to an existing system is the target of this quality attribute. The amount of modifications that are necessary to add a multilingual pattern to the existing system, is a quantified value. Ideally, multilingual pattern can be added to the system without any changes. So, the value would be 100. The amount of changes in the structure of system in comparison with other patterns changes, determine the value of this quality attribute.

2.8 SQ11: Encapsulation from developers

This quality is related to hiding pattern implementation and deployment from the programmers. In the worst case, all programmers should know about the issues of adding multilingual patterns to their codes, so the value would be 0. In the best case, the programmers add only one line of code in order to support multilingual feature in the code, which set the value to 100.

2.9 SQ12: Data Access Frameworks support

In modern programming world, there are many data access frameworks that help the developers of systems write agile codes. For example, Java Framework has JPA, Hibernate, EclipseLink, JOOQ, and .NET Framework has NHibernate and Entity Framework. These frameworks hide the details of SQL implementation from the view of programmers. Some multilingual patterns have been deployed in very detailed SQL implementation. Therefore, data access frameworks cannot support them. In this case, the pattern or the data access frameworks should be avoid. If the data access framework supports the multilingual pattern, the value of pattern would be set to 100, otherwise set to 0.

2.10 SQ13: Complexity of the pattern

The complexity of implementation and development of the pattern and then using it, is the target of this quality attribute. If implementing the pattern is easy, the value of this pattern is set to 100. The complexity of implementation and deployment decreases the above-mentioned value.

2.11 DQ1: DBMS support

Some multilingual patterns are not supported by some DBMSs. For example, some patterns use the Object Relational aspects; therefore, relational databases do not support them. The value of this pattern is set to 0 for the case without DBMS support and it is set to 100 for the case with DBMS support.

2.12 DQ2: Nullification and sparse tables

Some multilingual patterns are causing nullifications and spare tables in the database due to their architecture. This quality attribute shows the effect of quality of nullifications in the system and the value can be a number between 0 and 100. In the best case, if the pattern does not create or force nullification or sparse tables, the value is set to 100. The amount of nullifications and sparse tables decreases this value.

2.13 DQ3: Redundancy

Some multilingual patterns force developers to write the same data in different parts of the database which is called redundancy. The amount of pattern redundancy is the value of this quality attribute. In the best case, if the pattern does not cause any redundancy, the value is set to 100.

2.14 DQ4: Size of one table

Some multilingual patterns force developers to write many data values in one table. These patterns have to challenge with issues such as the ability of cashing, maintainability, backup and transactions. If the pattern does not create a big data table, the value of this pattern is 100, otherwise it is 0.

2.15 DQ5: Number of Tables

Some of the multilingual patterns increase the number of tables in the system. In the worst case, some of them double the number of tables. In this case, the value of pattern is set to 0. In the best case, with adding at most one table, the value is set to 100.

2.16 DQ6: Performance of Querying

Some multilingual patterns decrease the performance of querying in the system. To quantifying this patterns, the performance of system with and without applying the pattern should be measured. This pattern is generally related to four types of CRUD querying (Create, Retrieve, Update and Delete). In the best case, the pattern does not affect the performance; therefore, the value would be set to 100. The amount of performance reduction for each query, decreases the value. The value of this pattern can be a number between 0 and 100.

2.17 DQ7: Easy Querying

The complexity of queries are the matter with applying the multilingual patterns. For example, if applying the pattern causes using additional joins to retrieve data from database, the complexity will be increased. The value of this pattern can be a number between 0 and 100. In the base case, without adding any complexity to system, the value is set to 100.

2.18 DQ8: Default Language querying performance
In most multilingual systems, the number of default language retrievals is too many. So, in the best case, multilingual pattern should establish a mechanism that the performance of retrieving the default language is not affected. In that case, the value of this pattern is set to 100. The amount of performance reduction for default language, lowers the value of this pattern.

2.19 DQ9: Two Language querying performance
Similar to the previous case, in most multilingual systems, the system only works with two languages. So, multilingual pattern could have a different mechanism to retrieve two base languages and the performance would not be affected in this case.

2.20 DQ10: Backup and Restore just one language data
To support this quality attribute, the multilingual pattern should have a structure that developers and end-users can easily backup and restore the data of one language. In that case, the value of pattern is set to 100, otherwise 0.

2.21 DQ11: Normalization
Normalization is one of the important principles of data modeling and database design [6]. Some multilingual patterns violate this principle. So, if developers use that pattern in a non-professional manner, this may cause a global problem in the system. For example, cascading update or delete would be violated if the database design was not normal. Therefore, using the patterns require a professional team of database designers and programmers. The value of this pattern is a number between 0 and 100. In the best case (without violating normalization), the value of this pattern is set to 100.

2.22 Data Modeling Patterns to support Multilingual
In the following sections, all existing data modeling patterns have been explained and their main weaknesses and strengthens have been described. As the detailed reasons of assigning a number to each pattern are too many, so just for only the first pattern has been described. Notably, P5 is a new data modeling pattern that is given in this article.

2.23 P1: New column for each language for each field in the same table
This pattern is the simplest data modeling pattern for multilingual support. It creates an additional column for each language and for each field in a table if the field needs its translation in other language.

Suppose that a system has a table named News with four columns (fields): id, title, abstraction and body. The system works with the default language. Other languages have been asked to be added to system such as France (fr). So, each field in the News table that needs to be translated, should be duplicated. Therefore, the number of fields of News raise to seven. The final filed names of News are: id, title, title_fr, abstraction, abstraction_fr, body, and body_fr. For another language in the same News table, the number of translation fields should be duplicated again. In the following parts of the article, the above-mentioned quality attributes have been described and measured for P1.

(SQ1=60): To add a new language to the whole system, the structure of the table should be changed and a new translation field should be added for each field. In small systems it is easy. Like adding a new language, deleting a language from system or from one field in development time would be the same; so the values are: SQ3=60, SQ5=60, SQ7=60. (SQ2=0, SQ4=0, SQ6=0, SQ8=0): The new language cannot be added or deleted in running time. (SQ9=100): By adding new languages, the structure of source codes for the default language will not change. (SQ10=0): P1 cannot be added to a system without changes; so, modularity is zero. (SQ11=0): By adding a language to system, all source codes and as the same way, all programmers should be aware of the mechanism and source code will be hardly affected by the pattern. (SQ12=100): All data access frameworks support P1. (SQ13=100): P1 is the simplest patterns. (DQ1=100): All DBMSs support P1. (DQ2=70): Initially, there are columns for all used languages in the system. But some languages of the system may not be filled. (DQ3=100): There is no redundancy in P1. (DQ4=100): There is no giant table in P1. (DQ5=100): There is no increment in number of tables in P1. (DQ6=100): Because of reading from the same table without any joins, the performance of querying is not effected. (DQ7=90): There is no joins in P1, but the programmer should know about the name of fields. (DQ8=100): There is no changes in default language performance. (DQ9=100): All languages of a tables are in the same table, so querying two languages will perform in maximum performance. (DQ10=0): All languages of a tables are in the same table, so backup and restore cannot perform easily or automatically. (DQ11=60): In theory, all tables in multilingual mode in P1 are normal, but in practice, there is no language table that keep all languages with id and then use the id in other table for multilingual support.

2.24 P2: New row for each language in the same table with Language Id
This pattern is similar to P1, but instead of duplicating the content in columns, it duplicates in rows with a language id. For new table with four columns: Id, title, abstraction and body, new field named LangId should be added. So each row holds the data for specific language. To observe normalization, the id of table and LangId should be a primary key together. This pattern attempts to improve the weaknesses of P1 and make a new pattern that new languages can be automatically added even in runtime. But, in this case, some other side effects and weaknesses will be created. For example, if the data of a raw is duplicated, the whole date of the row will be duplicated as well. Some columns may not be multilingual. Therefore, if the programmer duplicates the data, redundancy will increase and if she does not duplicate the data, nullifications will be created.

2.25 P3: Single translation table for whole system
In this pattern, translation of all languages are stored in a single translation table. It is more suited for dynamic websites which have a large number of languages and intend to add a new language in the future at runtime. It is so modular and can be added to any system without changes in the system structure. One translation table should have relationship to all tables of the system to know about the reference field to be translated, which is impossible in DBMS, so these relationships should be checked in source codes. So, it

violates normalization and some data access frameworks do not support that. On the other hand all data stored in one table. This action affects all parts of database such as performance, maintainability, cash ability, etc. In this pattern, default language can be stored in main table and only the translation is stored in one table.

2.26 P4: One translation table for each table

This pattern is a variation of P3 and it seems to be easier to maintain and work with. An additional table is created for each table that stores information that may need translation. The original table stores only language insensitive data and the new on stores all translated data. If the original table stores all fields of table in default language, the performance and maintenance of the default language will be preserved. Translation table has a language Id, original table Id, and all fields of the original table that needs to be translated. In this pattern, retrieving other languages needs only one join in the database; therefore the performance slightly decreases.

In order to support the performance of two languages, the combination of P4 and P2 could be used. To do that, the default language together with the secondary language are stored in the original table and other languages are stored in translation table. In this case, system will have some redundancy or nullification. Instead, the performance of two language systems supports powerfully.

2.27 P5: Array Type for multilingual field in the same table for the multilingual field

For pages

In Object Oriented DBMSs and Object Relational DBMSs, the array type exists [7]. For example, the type of a filed can be set as array of String (String[]). So, if any field of a table needs to have multilingual support, the data type of that field can be set as array. In this case, the first index of the array of all multilingual fields are default language, and the second index is the second language, etc.

The most important weakness of this pattern is that only Object Relational DBMSs such as PostgreSQL and Oracle or Object Oriented DBMSs such as ObjectDB support this pattern. Apart from this disadvantage, this pattern is so extendible, modifiable, and powerful. There is no need for joins in retrieving data; and the performance of querying is not effected.

2.28 P6: Use DBMS Locale System

Some DBMSs have Locale System and support multi-locale databases. Before each query, programmers should set desired locale and then the result of query will be ready in that locale. DBMS hides the implementation of multilingual support and programmers only set the locale and get the date. Encapsulation and ease of use are the strengths of this pattern. On the other hand, Locale System does not support all DBMSs and in many DBMSs it will not work properly for all languages. This pattern is a black box and DBMS hides the details of implementation, so in order to use this pattern, all performance and maintainability and other database issues should be checked for each DBMS.

2.29 P7: Using a complete translation system as a middleware

The idea used in this pattern varies from other patterns. Instead of saving each data translation for each field in tables, all data of all languages are saved in a translation system. This system can be an Expert System that use some artificial intelligence techniques to save and retrieve data for all languages. For example, GLUE!-PS [8] architecture is one of them. This System can be a middleware between any system and its database and translate each data to desired language. Obviously, implementing this type of systems are so complex, but it can be developed as a powerful middleware or a plugin on DBMSs.

3. THE PRIMITIVE EVALUATION FRAMEWORK

In Table 1 and Table 2, the results of evaluating pattern by evaluation parameters are given. All patterns and their values for quality attributes are presented. Note that some quality attributes of some patterns cannot be measured. So, the value marked as '?'.

Table 1. Software Engineering related Quality Attributes

	SQ1	SQ2	SQ3	SQ4	SQ5	SQ6	SQ7	SQ8	SQ9	SQ10	SQ11	SQ12	SQ13
P1	60	0	60	0	60	0	60	0	100	0	0	100	100
P2	100	100	100	100	100	100	100	100	100	100	30	100	95
P3	100	100	100	100	100	100	100	100	100	100	30	30	95
P4	100	100	100	100	100	100	100	100	100	100	90	100	95
P5	100	100	100	100	100	100	100	100	100	100	100	100	100
P6	100	100	100	100	100	100	100	100	100	100	100	100	100
P7	100	100	100	100	100	100	100	100	100	100	100	?	?

Table 2. Database Systems related Quality Attributes

	DQ1	DQ2	DQ3	DQ4	DQ5	DQ6	DQ7	DQ8	DQ9	DQ10	DQ11
P1	100	70	100	100	100	100	90	100	100	0	60
P2	100	70	0	100	100	85	90	90	90	0	60
P3	100	100	100	0	100	50	50	100	50	100	0
P4	100	100	100	100	0	85	85	100	50	100	100
P5	0	100	100	100	100	100	100	100	100	0	100
P6	0	?	100	?	100	?	100	?	?	?	?
P7	?	100	100	100	100	?	100	?	?	100	?

4. CONCLUSION

Multilingual support is the main feature of Information Systems. There are many patterns with many related quality attributes to support multilingualism in databases. In this article, all existing pattern with all quality attributes have been categorized and explained. So, Information System developers and architects would be able to choose one of the provided patterns based on their quality attributes.

As future work, the quality attributes of Information Systems can be added. For example, the size of system, number of packages in the system and specially type of the system, such as Accounting System, Social System, etc. These parameters affect choosing a pattern. On the other hand, this article approaches to a building a framework. In order to complete the framework, a selection mechanism should be defined and added to this work, so that developers and architects can select appropriate pattern based on a mechanism.

5. ACKNOWLEDGMENTS

Special thanks to Dr. Rahim Dehkharghani for his adorable helps.

6. REFERENCES

[1] http://whatis.techtarget.com/definition/internationalization-I18N.

[2] Patrick A.V. Hall, Martyn A. Ould, eds. (1996). Software Without Frontiers: A multi-platform, multi-cultural, multi-nation approach. With contributions and leadership by Ray Hudson, Costas Spyropoulos, Timo Honkela et al. Wiley. ISBN 9780471969747.

[3] Michael R. McCaleb (1999). "A Conceptual Data Model of Datum Systems". National Institute of Standards and Technology. August 1999.

[4] "The Data Model Resource Book: Universal Patterns for Data Modeling" Len Silverstone & Paul Agnew (2008).

[5] Chen, Lianping (2013). "Characterizing Architecturally Significant Requirements". IEEE Software 30 (2): 38–45.

[6] Codd, E. F. (June 1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387.

[7] S. Sumathi, S. Esakkirajan, Fundamentals of Relational Database Management Systems, Springer, 2007.

[8] L. P. Prieto, J. I. Asensio-Pérez, Y. Dimitriadis, E. Gómez-Sánchez, J. A. Muñoz-Cristóbal, GLUE!-PS: A Multilingual Architecture and Data Model to Deploy TEL Designs to Multiple Learning Environments, Book Title: Towards Ubiquitous Learning, pp 285-298 , Springer Berlin Heidelberg, 2011.