Journal of Applied Sciences

ISSN 1812-5654

Journal of Applied Sciences 7 (1): 91-97, 2007
ISSN 1812-5654

(DynaDBEdit): Dynamic Database Textbox Field and Validator for Web and Application Software

[1]Yasar Guneri Sahin and [2]Halil Ibrahim Bulbul
[1]Department of Computer Engineering, Yasar University, Izmir, Turkey
[2]Department of Computer Education, Gazi University, Ankara, Turkey

Abstract: When the database tables are concerned, alterations in the structure of table attributes have a direct impact on the edit fields those are used in database application software (e.g., Web pages, application software). In case when any of the attributes are changed, it becomes necessary for software or Web pages to be recompiled following the changes made in edit fields. This process of recompilation is a long and costly operation, which means a temporary pause for transactions. For this reason, that it causes important loss in financial sector. Besides, the maxim of the software engineering is that the final product should require the least updating. This study offers an object and component that aims to remove the necessity for the recompilation of application software following the amendments made in the attributes. This component is a contribution to software development and maintenance for it is capable of automatically obtaining the data of domain for the attributes in database tables and controlling the data entrance process. Moreover, it is also a contribution for the users and practitioners in the sense that it reduces the time required to correct the errors and to ensure data coherency database integrity.

Key words: Field validator, software developing, domain builder, textbox

INTRODUCTION

Although, it is not a common practice to make changes in database table structures following the process of system integration of multi-tier application software and their maintenance, sometimes may be required. Additionally, domains determined for attributes require changes usually. Since, the changes, which became necessary due to some later information and that are originated from legislation, will directly affect the application tier (layer); it will necessitate the recompilation of application software. And, when the Web applications are concerned, these changes necessitate an update in the relevant areas of the Web page.

Such a recompilation in application software and an update in Web pages require a temporary pause in the operation of application software or for Web pages to go temporarily offline. It simultaneously requires a redistribution of application software to clients when the intranet environment is concerned. Interruption in the operation of application software could result in great losses especially in the finance (or banking) sector and when this application software is used in environments having direct link to production line. In addition to that, sending of false data during the integrity constraints (type constraints) and domain check operations would result in an increase in data traffic and an increase in the amount of time spent for the re-sending of true data. This study offers a method aiming at a contribution for ensuring database coherency and avoiding any cut offs in the operation of application software due to the changes made to relational attributes. It would in this way be possible to prevent great loses of time and money.

MOTIVATION AND RELATED WORKS

The maintenance phase, as one of the most important elements of software development process, is an open-ended process. The necessary changes that would arise at this phase would affect the permanence and continuance of the project and would result in higher costs. Therefore, adding in advance the changes to the software that would be required in the future would lay the burden of users and practitioners down. From this point of view it could be said that defining variable and dynamic edit fields would be very useful during the application phase. The content of the method presented in this study is obtained through a consideration of possible necessary changes that could arise in all kinds of software in the later phases of application. It is quite possible to make additions to the list of those changes.

Corresponding Author: Yasar Guneri Sahin, Department of Computer Engineering, Yasar University Borova Izmir, Turkey
Tel: +90 (232) 461 4111/305 Fax: +90 (232) 461 4121

Possible type alteration in attributes: The changes that could be made in the structure of attributes could be examined as followings:

- In this kind of change a pre-defined type is changed with another type. A change in the composition of defined account numbers by a bank from digits into characters could be an example to this kind of change. Another example could be given from an integrated meat facility where selling unit shown in the stock table is integer since the cattle is sold undetached. Changing of this unit into kilo would double the area, is another example for type changes. Such cases would require all related field types in the application layer to change.

Such kind of changes arouse as a response to an increase in the amount of data entered in time. A bank's increasing its customer number format from 7 (###.$$$.##: where # stands for numbers and $ for characters) to 8 digits (##.$$$$.##). This situation requires an increase in the number of digits in entrance as to make it comply with 8 digits and thus a recompilation. Many other examples could also be given for this situation.

Possible attribute domain alterations: Despite being similar to constraint changes, domain changes that is frequent type of change, have some differences with constraint changes with respect to numerical fields. This kind of change is made in the monetary attribute domain especially due to the currency changes. Such kinds of changes affect almost all commercial sectors and would result in important financial and time loss. An important example for this is the Turkish experience of eliminating six zeros from in the new Turkish currency, during the conversion from TL to YTL. Following the conversion, 1 million TL became equal to 1 YTL and it required all data entrance intervals and data entrance masks to change in every field related to monetary affairs. It is also possible to give various examples for the change of domain situations.

The changes that are mentioned above are of the sort that could be encountered in almost all applications. It is therefore very important to prevent monetary and time loses that arouse as a result of the recompilation processes which are caused by the above-mentioned changes. This study would present the DynaDBEdit (Dynamic Database Field Editor) as a method that could be employed to prevent recompilation loses.

Related works: Despite there are some studies on methods of updating database structure, there is no general study that deal with all data types that are capable of being fully editable. Additionally, the studies carried out in this field were usually theoretical works that had underestimated the data operations carried out in the layer of application. This section of the present study mentions some studies that were carried out in this field.

One of the studies was carried out in a bid to develop the update and operation features of multimedia environments over a database (Kambur *et al.*, 2003). The study presents a query update method that is based on programming languages. However, the study is only related to multimedia environments.

Ducrou, Wormuth and Eklund's study are dealing with the construction of relational schema navigation by using formal concept analysis (Ducrou *et al.*, 2005). In that study a method was developed in Web based applications and was accompanied by implementation of the method. That study is akin to this study with respect to obtaining work schema information. However, the basic difference with this present study and of Ducrou, Wormuth and Eklund is that their work is just attempts of making analyses.

Hilderman, Hamilton and Cercone dealt with data mining over databases through using domain generalization graphs in their study. Moreover, this study was resonated with its implementation on Serial versions and parallel versions of the Multi-attribute generalization algorithm for multiple-attributes (Hilderman *et al.*, 1999). Despite being akin to this study with respect to obtaining attribute domains, Hilderman *et al.* (1990) study is more focused on data mining. Method and components of this study were completely focused on application and was brought to a state where it can be used with all databases. In another study, a method was introduced in order to decrease or solve the data conversion problems by constructing a new domain introduction layer within the database by using MesoData (DeVries and Roddick, 2004).

Some other researches and works that could somewhat be related to the issue were also examined. Some of those studies were about software developing and productivity (Fritsch and Renz, 2005; Raghavan, 2002). The present study dealt with some other related works with respect to method and application (Chua *et al.*, 2003; Mungnirun, 2006; Oommen and Thiyagarajah, 1999; Gunopulos *et al.*, 2005; Weber, 2002; Castano, 1998; Hull, 1986; Miller, 1994; Sjoberg, 1993; Steidl, 2001; Cheung *et al.*, 2005; Guruge and Stonier, 2006; Magkanaraki *et al.*, 2002). Despite being a valuable source for the present study, these other studies differ with the present study with respect to application and methodology. The main difference is that those studies were carried out on meta-data applications and schema but a general pattern of application was not given. On the other hand, there is a similar text box component that is developed by Active Up Company over ASP. net. Despite this component allows to enter mask information and type information, it doesn't allow for domain control and dynamic updating. The method introduced in the present study would allow adding dynamic updating and domain control functions to text box component (Active Up, 2006).

Dyna DBEdit engine: The aim of this study is to prevent interruption of the application software mentioned in the introduction section and to allow required changes in any moment with a contribution to database integrity and consistency. In order to reach this aim, the study introduces a method and based upon this method develops an object and a component.

DynaDBEdit engine, the general structure of which is shown in Fig. 1, works once in each loading of the software. By controlling the attributes in the relevant relations it transfers the necessary changes to the relevant areas the application form. This object that is directly connected to the application layer which is located in data module, works as a bridge between database and form edit fields. Data that is sent and received between edit fields and database would be filtered with this object in order to prevent transfer of wrong data. In this way, it would both contribute to prevent errors that could be done while editing data and a contribution to the coherency of the database.

DynaDBEdit object: The DynaDBEdit object that is developed for database connected application software is composed of three main parts. Figure 2, shows a schema of the DynaDBEdit object. Although changing with respect to application development environments, the order of these three main parts is the constructor, the event handler and the destructor.

DynaDBEdit constructor: The constructor part, which is the main part of the object, is itself composed of three parts that are field type reader, domain checker and constraint builder.

The constructor automatically locates itself on the create event handler (onCreate, frmBuild) of the form that it is located on. Initially it reads the relational schema to which the component is connected. Following the process of schema reading it determines the field type from the properties of the attribute it is diverted to. The field types that could be used by the object are given in Table 1. All editable general types that could be defined in databases are added to this object.

Following the type reading and object type defining process, the process of editing field domain identification is initiated. The process of domain identification changes with respect to the objects [domain-identifier] property. Domain Identification process is carried out by

Table 1: Acceptable attributes types by DynaDBEdit

Numerical types		Character types	
ID	Type	ID	Type
tInt	Tiny integer	bStr	Binary string
sInt	Small integer	Char	Char
Int	Integer	varChar	Variable char
bInt	Big integer	longvarChar	Long variable char
utInt	Unsigned tiny int.	wChar	Wide char
UsInt	Unsigned small int.	varwChar	Variable wide char
Uint	Unsigned integer	longvarwChar	Long var. Wide char
ubInt	Unsigned big integer	varChar2	Variable char
single	Single	edit	Edit string
Double	Double	memo	Memo string
currency	Currency	text	Text strnig
decimal	Decimal		
numeric	Generic numerical		
boolean	Boolean		
date	Date		
time	Time		
varnumeric	Variable numerical]		

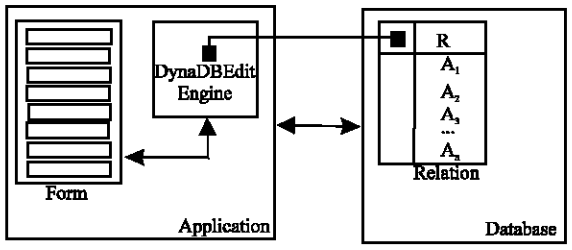


Fig. 1: General structure of DynaDBEdit

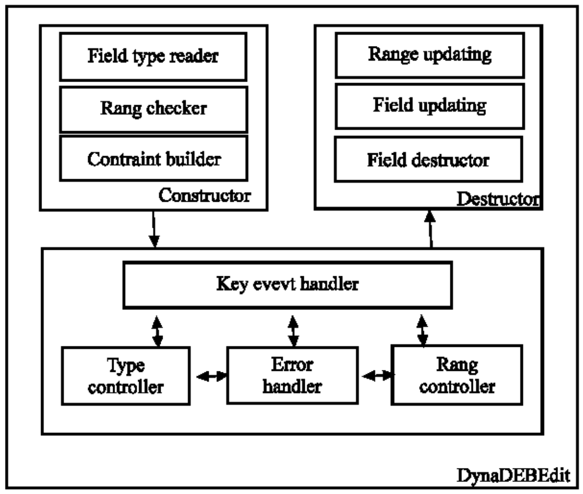


Fig. 2: DynaDBEdit object schema

Table 2: Default domains for di_default

	Attribute max			
Type	Size	Size	Value interval	Characters permitted
Character	1	1	-	#0..#255 (except special chars)
String	x	x	-	#0..#255 (except special chars)
Integer	-	9	-2147483648..2147483647	0..9,-
Real (float)	-	12	$2.9\times10^{-39}..1.7\text{x}10^{38}$	0..9,-,.,,,Ee
Date	10	10	01.01.0001..31.12.3000	0..9,-,.,,,Ee

employing the method chosen in accordance with the demand of user.

Domain_identifier = [di_none, di_default, di_intellegent];

If the method chosen among the methods mentioned above is di_none, then the domain is identified as to allow negative and decimal characters on the condition that no space will be given between characters and characters to be used for alpha-numerical fields, numbers for numerical fields.

If the di_default value is identified as domain-identifier, then the maximum limits that are allowed by the field type would be used for domain. In other words, if a 30 character string type is given to the field than a domain composed of all the 30 characters would be used (in case that the NOT NULL condition isn't determined in the schema) for that field. Table 2 shows the domains that are used for default domain (di_default). The domains for the sub-values that are not listed on the Table would be applied in accordance with the properties of the database.

If the Domain-identifier property is identified with the value of di-intellegent, then a new domain would be determined by using the information obtained following the control of pre-stored data. If the field in question is a numeric type than the smallest and greatest numerical values entered in that field will be determined by running SQL sentence like the below one:

```
Select min (attribute name), max (attribute name) From
table-name
```

The minimum and maximum values that are obtained through the queries would be used as domain for that field. The digital value that is entered by the application software user is controlled by the domain controller which is in the event handler section by using the identified domain. The values that are not between the intervals will be identified as error and an error message would be given on_exit event. However, in some cases (especially at the initial stage of data editing to the tables) it is possible to edit values out of the range of domain without any error message. The application asks for acceptance for each value that lies out of the range of domain. As a result of this acceptance, the domain will be changed as to include the value edited. Following this operation of domain change, the new set will be saved in the memory address that is same with the local address of domains. Sending the domain changes to all clients and updating the memories in the Internet or intranet environments will be carried out with a data dictionary work over the database.

If the field in question is an alpha-numeric, then the domain identification process would show differences through numerical fields. In this case, initially the character size of the attribute will be determined by obtaining the table schema. Then, if the application is executed for the first time after this process than the operation to be done is to constitute field character or string pool. The pool, which will include all characters or strings that could be edited to that field, is located in a memory that is belonging to that field. In order to determine the pool, a random reading of a 300 string will be used for sampling. And, the character pool that is obtained as a result of sampling will be used for the evaluation of new data that will be entered.

Human name edit fields' domain could be given as an example of this pool system. It will be possible in this way to prevent wrong spelled names by gathering the unique names in the pool. Entering spelled wrong names is a very common mistake. The wrong entering of the name Michael as Micheal could be seen as a simple misspelling of a word but nonetheless it could cause very important problems. Preventing such misspellings would be very important both for decreasing the database traffic (since errors requires fixation) and increasing the consistency of the database.

What is important here is to apply constraints as successful as it can be. This simple system that is used for constituting character domain could be made into a more useful system by supporting it with developed neuron-network applications. Error control system that is used in numerical fields could also be used here to prevent entrance of erroneous characters by informing with error messages. If the error message is still given despite there is no erroneous entrance, then the characters that is perceived by the system would be added to the character pool in order to prevent further error messages.

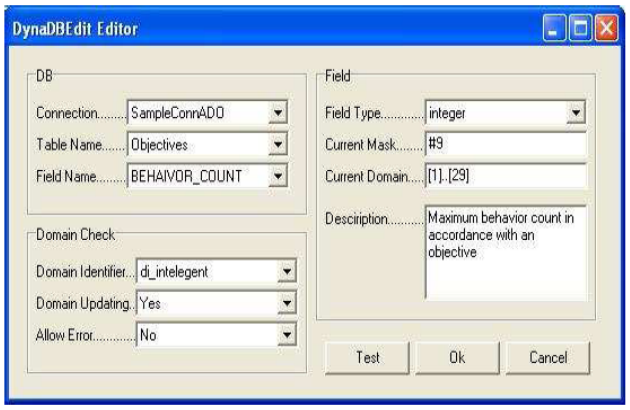
Event handler: Another part of the object is event handler. This part is composed of three parts which are dependent on key event handler unit. In this part key event handler, firstly sends the character of the pressed button to the type controller by controlling each key event pressed on the keyboard and provides type sensitivity to that character to be controlled. If there is an error in the results of control process, error handler causes an internal exception error. Having been evaluated by key event handler, this error is sent to the field. If the error is a fatal error like an alpha-numerical character pressed in numerical fields, key handler without sending an error message or an information message to the user, causes the pressed key to be zero and to be deleted from the. In other words, the value of the key pressed by the user is not written to the field, thus preventing entry of the character, the field is provided to be filled orderly field (this working is same with mask-edit field developed by Borland, Microsoft, etc.,.

To exit from the field, the system calls the on-exit event. This event provides the domain controller to operate automatically and the data edited to the field to be controlled by the domain controller. If an error occurs at the result of the control, the error message is sent to the user by the error handler. When the user closes the message appeared, control is again passed to key event handler unit in order to correct the erroneous field. If the user declares that the entered value is not false when error message appeared, an automatic approval message is sent by the object. If this approval message is confirmed, through including the value which is entered new and did not take place in domain before, the object provides the control to pass to destructor unit for its exit from the field.

Destructor: This part is the last part which is used in the course of exit from the field. Field destructor unit which is one of the three units of this part is used to notify the object of the closure of the field and to destruct the places separated for the object in the memory. Then, field updating unit provides entry of the information to the necessary part of the database. Lastly, domain updating unit, by renewing the domain in accordance with the last edited value, stores new domain information in the appropriate memory area.

DynaDBEdit component: DynaDBEdit object is developed as to allow identification and use in different application development tools by means of a dynamic link library. DynaDBEdit component is a component developed on Delphi, C++, VB, C# application and development means by making use of this library.

The component can also use data dictionary information. The component performs database connection by making use of connection objects which take place on data module dependent on form. Figure 3. indicates the type editor belonging to the component. After the component is placed on the form, by using the editor it is possible to make changes on database connection and field information. Characteristics of the component (owing to characteristics of the object) can also be changed in run-time. Thus, in case of an online change that can be made on the field, the component can be updated without interrupting operation of form.


DynaDBEdit Editor

DB
Connection......... SampleConnADO
Table Name....... Objectives
Field Name......... BEHAIVOR_COUNT

Domain Check
Domain Identifier... di_intelegent
Domain Updating.. Yes
Allow Error............ No

Field
Field Type............ integer
Current Mask........ #9
Current Domain..... [1]..[29]
Desciription........... Maximum behavior count in accordance with an objective

Fig. 3: Field property editor of DynaDBEdit component

RESULTS

Here, data and experimental results obtained during the test of the application software developed with the use of component. Table 3 shows the test environment variables.

Test software is executed without an edit field domain control (di-none) and by using normal control (di-default) and the developed method (di-intellegent). Fig. 4.

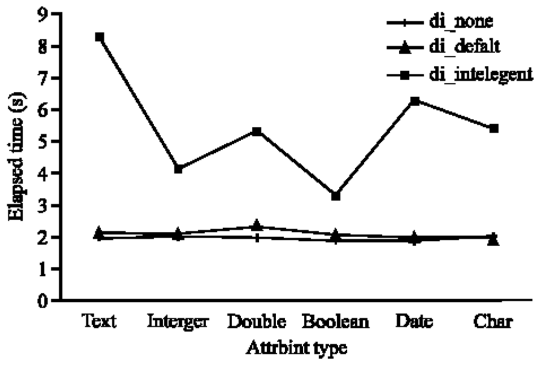


Fig. 4: Average connection time in accordance with data types

Table 3: Test environment and descriptions

Test environment		Description
Hardware		
	Server	IBM Pentium IV 3 GHz
		15xIBM 8424-7BG Think
	Clients	
Software		Centre Pentium IV 3G
		Windows 2003 Server, Windows
	Operating system	
		XP Professional
	DBMS	Microsoft SQL Server 2000
	R.A.D.	Borland Delphi
	DB connection tool	ADO
	DB	Student Database
Data	Total number of tables in	
	DB	18
	Total tuple count	360 000
	Avg. attribute count in a	
	Table	16
	Test filed types	Text, Integer, Double, Boolean,
		Date, Char
Test	Connection retry count	40
	Total count of record	
	entered	9 000 (600*15)
	Number of proper entrance	8 100 (90%)
	Number of improper entrance	900 (10%)

Table 4: Results of proper and improper data input test

Number of record	Field correction duration (normal)*	Field correction duration (dynadbedit)	Difference (%)
8 100 (proper)	192 min	196 min	-2
900 (improper)	69 min	24 min	187.5
9 000 (total)	261 min	220 min	18.6*

di_none

indicates average time spent for the necessary controls carried out in accordance with the fields during the establishment of database connection at the initial moment of execution.

The connection elapsed times indicated in Fig. 4. with the developed method, are approximately three times more than others. However, in this condition, time consumption occur only one and in the first moments of the execution process of the software and running period does not affect this time so much. Although use of the component seems time consuming, because actual time spend in application software occurs during run-time, this time consumption is in a level that can be ignored.

Table 4 indicates the results of 8100 proper, 900 improper, a total of 9000 entry inserting and updating test process performed on the fields by 15 person test team.

In Table 4, in the conditions that DynaDBEdit component is not used in application software and proper entries are edited to the domain, there is not a gain of time. Nevertheless, in case of improper data edition or correction, there is, %187.6 saving in time because of the duration needed for entry rejection by the software and reaction lengths of the users to approve and detect the errors. Total time saving is calculated as %18.6. Database domain errors occurred in the course of data entrance will be removed and motivation of the users will be affected in a positive manner.

CONCLUSIONS

This study, which is made in addition to the studies made on database applications, aims to prevent the necessity to recollect application software due to the change that will be made on attributes of tables. In addition, in order to ensure data coherency, to maintain database integrity and to avert time loss due to correction of the errors, an edit field (TextBox) object and component is presented.

It is observed that use of DynaDBEdit made easier the control of the data edited in application software edit fields by the users. Furthermore, prevention of false value entrance of data is a significant gain in terms of system integrity and consistency. There has been a relative improvement and acceleration in data entrance speed and through this object which can be used by application developers, application and improvement process will get easier.

Updating the domains and renewing server information in order for DynaDBEdit field to be used in distributed databases can be planned as a future study. Easy use of the information belonging to the field and data through developed data dictionary can be though as a future study as well.

REFERENCES

Active, Up, 2006. Masked TextBox. http://www. activeup. com/products/components/activeinput/maskedtext box. aspx

Castano, S., 1998. Conceptual schema analysis: Techniques and applications. Acm transactions on database systems, pp: 23-286.

Cheung, K.H., K.Y. Yip, A. Smith, R. deKnikker, A. Masiar and M. Gerstein, 2005. YeastHub: A semantic web use case for integrating data in the life sciences domain. Bioinformatics, 21: 185-196.

Chua, C.E.H., R.H.L. Chiang and E.P. Lim, 2003. Instance-based attribute identification in database integration. The VLDB J., 12: 228-243.

De Vries, D. and J.F. Roddick, 2004. Facilitating Database Attribute Domain Evolution Using Mesodata, Wang S. *et al.* (Eds.): ER Workshops 2004, Lecture Notes in Computer Sci. Springer-Verlag Berlin Heidelberg, pp: 429-440.

Ducrou, J., B. Wormuth and P. Eklund, 2005. Dynamic Schema Navigation Using Formal Concept Analysis. DaWaK 2005, Lecture Notes in Computer Science. Springer Verlag Berlin Heidelberg, pp: 398-407.

Fritsch, C. and B. Renz, 2005. Four mechanisms for adaptable systems a meta-level approach to building a software product line. Software Process Improvement Practice, 10: 103-124.

Gunopulos, D., G. Kollios, V.J. Tsotras and C. Domeniconi, 2005. Selectivity estimators for multidimensional range queries over real attributes. The VLDB J., 14: 137-154.

Guruge, D.B. and R.J. Stonier, 2006. Intelligent document filter for the Internet, Data Mining: Theory, Methodology Tecniques, ans Applications, Lecture Notes in Artificial Intelegence vol. 3755. Springer-Verlag Berlin Heidelberg, pp: 161-175.

Hilderman, R.J., H.J. Hamilton and N. Cercone, 1999. Data mining in large databases using domain generalization graphs. J. Intelligent Inform. Sys., 13: 195-234.

Hull, R., 1986. Relative information capacity of simple relational database schemata. Siam J. Computing, 15: 856.

Kambur, D., D. Becarevic and M. Roantree, 2003. An Object Model Interface for Supporting Method Storage. In Proc. 7th East European Conf. Advances in Databases and Information Systems ADBIS.

Magkanaraki, A., S. Alexaki, V. Christophides and D. Plexousakis, 2002. Benchmarking RDF schemas for the Semantic Web. Semantic Web-ISWC, Lecture Notes in Computer Science Vol.2342. Springer-Verlag Berlin Heidelberg, pp: 132-146.

Miller, R.J., 1994. Schema quivalence: In heterogeneous systems-bridging theory and practice. Inform. Sys., pp: 19-3.

Mungnirun, K., 2006. NPA database filter guide. Information system for neuronal pattern analysis. http://soma.npa.uiuc.edu/isnpa/filter/filter.html.

Oommen, B.J. and M. Thiyagarajah, 1999. On Benchmarking Attribute Cardinality Maps for Database Systems Using the TPC-D Specifications. Bench-Capon, T., G . Soda and A.M. Tjoa (Eds.): DEXA'99, Lecture Notes in Computer Science Vol.1677. Springer-Verlag Berlin Heidelberg, pp: 292-301.

Raghavan, G., 2002. Improving Software Quality in Product Families through Systematic Reengineering. Kontio J. and R. Conradi (Eds.): ECSQ, 2002, Lecture Notes in Computer Science Vol. 2349. Springer-Verlag Berlin Heidelberg, pp: 90-99.

Sjoberg, D., 1993. Quantifying Schema Evolution. Information and Software Technol., pp: 35-35.

Steidl, J., 2001. Downhole data dictionary and data formatting requirements. Invited Workshop on Archiving and Web Dissemination of Geotechnical Data, October, pp: 169-188.

Weber, G., 2002. Semantics of form-oriented analysis. Ph.D Thesis. Am. Fachbereich Mathematik und Informatik der Freien Universitat Berlin.