International Journal of Engineering & Technology, 7 (4.32) (2018) 1-6

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Automatic Lexical Alignment between Relational Database and Heterogeneous Big Data Based on NOSQL Systems

IBN BATOUTA Zouhair[1]*, HAJOUI Omar[1], TALEA Mohamed[1]

[1] *Hassan II University, Faculty of Science Ben M'Sik, LTI Laboratory, MOROCCO*
**Corresponding author E-mail: zouhair.ibnbatouta@gmail.com*

Abstract

In the absence of a universal consensus governing the use of models, the IDM has led to the creation of a large number of heterogeneous (distinct) meta-model systems with similar or complementary uses and objectives. To solve this problem of increasing heterogeneity of meta-models, we proposed an approach that we named generative automatic matching GAM. In this approach, we have dealt with the problem of the heterogeneity of meta-models in a new way that uses automatic matching, the alignments found are then took into profit to facilitate generation between source and target models which are conform to the linked meta-models.

This article presents an application of our GAM approach on a case study composed of heterogeneous meta-models of relational databases and big data, we specially treat the application of lexical automatic matching based on hybrid meta-heuristic. We have selected three types of databases and big data based on NoSql: Key Store, Document Store and Columnar Store; at the end of our article we present an evaluation of the results found based on quality mathematical measurements.

Keywords: *: Automatic Matching; Big Data; Columnar Store; Document Store; Generative automatic matching; Heterogeneous Meta-models; hybrid meta-heuristic; Key value Store; NoSql; Quality mathematical measurements.Relational;*

1. Introduction

Although model driven engineering has greatly propelled software engineering, our TSMR review and our multi-criteria comparative study of generation and automation approaches in software engineering [1, 2], allowed us to detect that the absence of a universal consensus governing the creation of models has led to the emergence of a large number of systems based on heterogeneous meta-models, with similar or complementary uses and objectives, leading to a problem of increasing heterogeneity of created meta-models.

In this context, we have proposed in [3, 4] the architecture and implementation of our approach called GAM (generative automatic matching), which is particularly useful as a palliative measure to this major disadvantage. This approach, which consists in combining the automatic matching of meta-models with the generation of models is, in fact, new and novel in the literature. Among other things, it will make it possible to overcome the many existing gaps in matching approaches [5, 6, 7, 8, 9, 10, 11, and 12].

In this article, we will apply GAM's automatic lexical matching on a case study composed of heterogeneous meta-models of relational databases and big data, the ultimate goal is to test the results of the lexical heuristics used in our GAM approach on a promising and in full expansion domain, namely big data systems; first of all, we will present a reminder of our GAM approach in section 2, then in section 3 we continue with our Big Data Meta-models (BDM) case study, after that, in section 4 we will sum up the results of the application of different hybrid lexical heuristic of our GAM approach on the case study, in section 5 we will end up with the conclusion and future works.

2. Recall: Generative Automatic Matching Approach (Gam)

In this section, we will present a reminder of our GAM approach. First of all, we will present the architecture of GAM as well as the matching meta-model MMG (meta-model of generative matching). The latter helps to identify the basic concepts treated by our approach, for example, the various elements of a meta-model as well as the relationship between them, the types of connections or possible alignments between two or more elements of the different meta-models, the management of the matching versions and also the matching history.

Fig.1 shows the overall architecture of our approach.

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

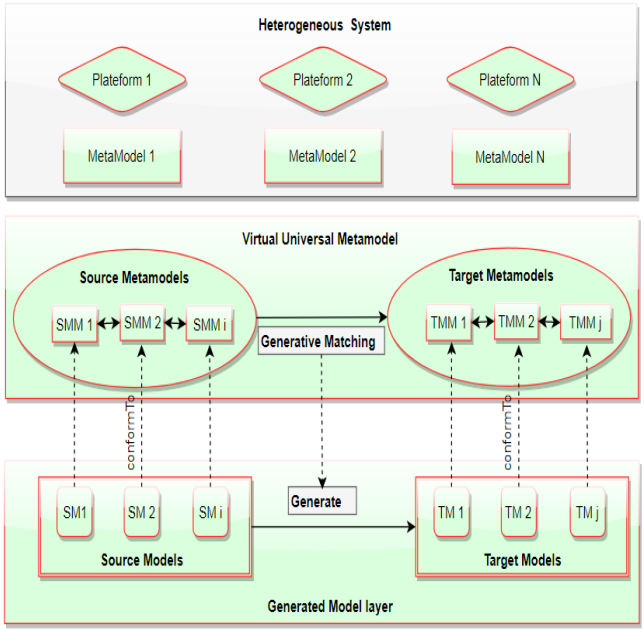

Fig 1: Global architecture of the generative matching approach

As shown in Fig.2, the set of platforms are seen as a heterogeneous global system, consisting of exogenous meta-models representing various domains.

The first step of the approach is to constitute a universal virtual meta-model consisting of source meta-models SMM1 ... SMMi, which will be matched with TMM1 ... TMMj target meta-models using hybrid heuristics. The resulting matching will allow descending to layer 1; Indeed, the approach will allow, from source input models SM1 .. SMi conform to the source meta-models, to generate target models TM1 .. TMj conform to the target meta-models.

Fig.2 introduces key notions of the meta-model MMG (Generative Matching Meta-Model):

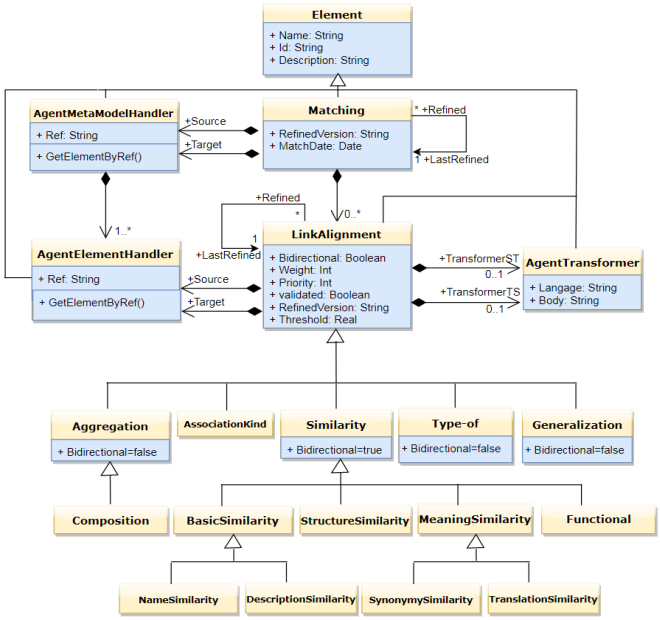

Fig 2: Core of the Generative Matching Meta-Model

In section 3, we will present our BDM case study of meta-models compliant to the MMG.

3. Case Study: Bdm

To test our GAM approach, especially the meta-models' automatic lexical matching part, we have designed the Big Data Meta-models (BDM) case study, which represents a heterogeneous database system, in fact we have developed four meta-models which represent four important types of database, namely relational databases and three types of NoSql big data [13, 14]: Key Value Store, Document Store and Columnar Store.
Fig.3 represents the source meta-model that is a relational SQL database.

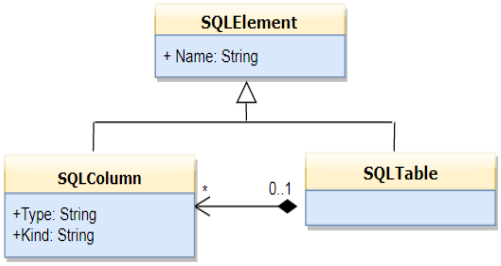

Fig 3: Source meta-model: Relational Database

Fig.4, Fig.5 and Fig.6 respectively represent the meta-models of the Big Data Key Value Store, Columnar Store and Document Store.

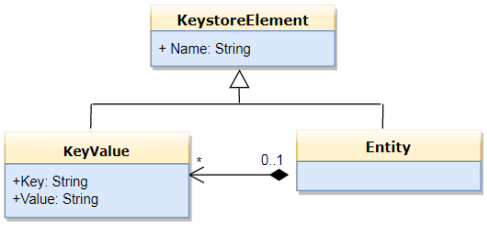

Fig 4: Target meta-model 1: Key Value Store

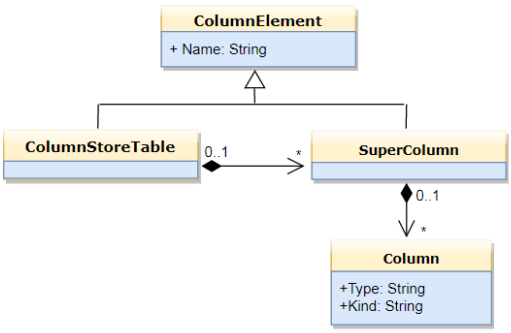

Fig 5: Target meta-model 2: Columnar Store

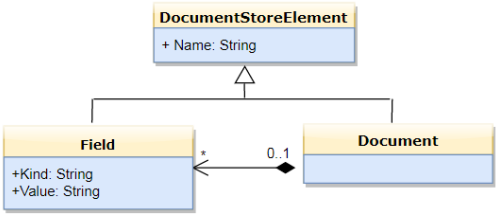

Fig 6: Target meta-model 3: Document Store

In the next section we will present the various automatic lexical matching heuristics applied to our case study, namely Name-Matching hybrid meta-heuristic.

4. Automatic Lexical Matching Results

In this section, we will present the lexical heuristic named Name-Matching, it allows the generation of lexical alignments between the elements of the meta-models; we will also present the results of its application on the BDM case study.

4.1 Namematching meta-heuristic

The nameMatching heuristic aims calculating the lexical similarity between two elements of MMG by comparing their names, this hybrid meta-heuristic is based on two sub heuristics:

• PSN heuristic (name similarity permutations): this heuristic takes a value between 0 and 1; it is equal to 1 if the names of two elements are composed of the same permuted words, for example: “superElement” and “Elementsuper”, the value tends to 0 when the two elements differ too much.

• The heuristic LSN (Levenshtein Similarity of Names): this heuristic compares the two elements based on a metric called the Levenshtein distance: the Levenshtein distance between two labels is given by the minimum number of operations needed to transform a label into the other, where an operation is an insertion, deletion or substitution of a character.

In Table 1 and Table 2 we recall the Algorithms which implement the PSN and LSN heuristics:

Table 1: Computing Permutation name similarity PSN

```
real namePermutationSimilarity(s1 string, s2 string)

1. HashSet  Split1 = Split(s1.toLowerCase())

2. HashSet Split2 = Split(s2.toLowerCase())

3. int lenght = Split1.size() + Split2.size()

4. string Union = Split1.computeUnion(Split2)

5. int Intersection = Union.size();

6. return intersection*2/ lenght;
```

Table 2: Recursive Computing Levenshtein Similarity of names RLS

```
int nameLevenshteinSimilarity(s1 string, length_s1 int, s2 string, int lenght_s2))

int cost

HashSet  Split1 = split(s1.toLowerCase())

HashSet Split2 = split(s2.toLowerCase())

 /* base case: empty strings */

 if (length_s1 = 0) return length_s2

 if (length_s2 = 0) return length_s1

 /* test if last characters of the strings match */

 if (split1[length_s1-1] = split2[length_s2-1])      cost = 0

 else cost = 1

/* return minimum of delete char from s1, delete char from s2, and delete char from both */

 return minimum(nameLevenshteinSimilarity(s1, length_s1 - 1, s2, length_s2    ) + 1,

                nameLevenshteinSimilarity (s, len_s    , t, len_t - 1) + 1,

                nameLevenshteinSimilarity (s, len_s - 1, t, len_t - 1) + cost);
```

To normalize the heuristic of levenshtein we applied the following formula which allows having a final value between 0 and 1:

$$\mathbf{LevNameSimilarity = 1 - lev(element1, element2)/max(len(element1), len(element2))}$$

To compute the final hybrid nameMatching similarity, we first apply the PSN heuristic, if the value is greater than a threshold “Threshold1”, we take the similarity value found and consider it as the final similarity, otherwise we apply a linear interpolation using two weights w1 and w2 allocated respectively to the heuristics PSN and LSN :

$$\mathbf{nameMatching = w1 * PSN + w2 * LSN}$$

Where $\mathbf{w1 + w2 = 1}$

For further calculations, we took the values w1 = 0, w2 = 1 and Threshold1=1.

4.2 Namematching results between sql and key value metamodels

The application of the NameMatching heuristic on the meta-models, source SQL and Target Key Value, gave the following results (Table 3)

Table 3: Lexical heuristic NameMatching results between SQL and Key Value Store

SQL Elements	Lexical NameMatching Value	Key Value Elements
Kind	0,1666667	Entity
Kind	0,25	Key
Kind	0,1333333	KeyStoreElement
Kind	0,125	KeyValue
Kind	0,3333333	String
Name	0,1333333	KeyStoreElement
Name	0,25	KeyValue
Name	1	Name
Name	0,4	Value
SQLColumn	0,2666667	KeyStoreElement
SQLColumn	0,2222222	KeyValue
SQLColumn	0,1111111	Name
SQLColumn	0,1111111	String
SQLColumn	0,2222222	Value
SQLElement	0,5333334	KeyStoreElement
SQLElement	0,1	KeyValue
SQLElement	0,2	Name
SQLElement	0,2	String
SQLElement	0,2	Value
SQLTable	0,2	KeyStoreElement
SQLTable	0,25	KeyValue
SQLTable	0,25	Name
SQLTable	0,125	String
SQLTable	0,25	Value
String	0,1666667	Entity
String	0,2666667	KeyStoreElement
String	1	String
Type	0,1333333	KeyStoreElement
Type	0,25	KeyValue
Type	0,25	Name
Type	0,2	Value

SQLElement	0,1	Entity
SQLElement	0,1	Key

In order to refine the results found, we will apply the threshold of 0.4 to the matching values found, Tab.4 illustrates the remaining matching.

Table 4: NameMatching results SQL/ Key Value - Threshold :0.4

SQL ElemEnts	Lexical NameMatching Value	Key Value Elements
Name	1	Name
SQLElement	0,5333334	KeyStoreElement
String	1	String

Table 5 shows all the real matching that exist between the two meta-models SQL and Key Value, this table of real values will allow us in section 5 to calculate the different mathematical metrics that will determine the precision of automatic matching heuristics.

Table 5: SQL/ Key Value - All real matches

SQL Elements	Key Value Elements
SQLElement	KeyValueElement
SQLTable	Entity
SQLcolumn	KeyValue
String	String
Name	Name

4.3 NameMatching results between SQL and Columnar Store Metamodels

Table 6 shows all the results of the application of the NameMatching heuristic on both meta-models: source SQL and target Columnar Store.

Table 6. Lexical heuristic NameMatching results between SQL and Columnar Store

SQL Elements	Lexical NameMatching Value	Columnar Store Elements
Kind	0,07692308	ColumnElement
Kind	0,0625	ColumnStoreTable
Kind	1	Kind
Kind	0,3333333	String
Name	0,1666667	Column
Name	0,1538462	ColumnElement
Name	0,125	ColumnStoreTable
Name	1	Name
Name	0,09090909	SuperColumn
Name	0,25	Type
SQLColumn	0,6666667	Column
SQLColumn	0,2307692	ColumnElement
SQLColumn	0,1875	ColumnStoreTable
SQLColumn	0,1111111	Name
SQLColumn	0,1111111	String
SQLColumn	0,6363636	SuperColumn
SQLElement	0,3	Column
SQLElement	0,5384616	ColumnElement
SQLElement	0,0625	ColumnStoreTable
SQLElement	0,1	Kind
SQLElement	0,2	Name
SQLElement	0,2	String
SQLElement	0,1818182	SuperColumn
SQLElement	0,1	Type
SQLTable	0,1538462	ColumnElement
SQLTable	0,375	ColumnStoreTable
SQLTable	0,25	Name
SQLTable	0,125	String
SQLTable	0,1818182	SuperColumn
SQLTable	0,25	Type
String	0,07692308	ColumnElement
String	0,1875	ColumnStoreTable
String	0,3333333	Kind
String	1	String
String	0,1818182	SuperColumn
Type	0,07692308	ColumnElement
Type	0,125	ColumnStoreTable
Type	0,25	Name
Type	0,1818182	SuperColumn
Type	1	Type

In order to refine the results found, we will apply the threshold of 0.4 to the matching values found, Table 7 shows the remaining results.

Table 7: NameMatching results SQL/ Columnar - Threshold :0.4

SQL Elements	Lexical NameMatching Value	Columnar Store Elements
Kind	1	Kind
Name	1	Name
SQLColumn	0,6666667	Column
SQLColumn	0,6363636	SuperColumn
SQLElement	0,5384616	ColumnElement
String	1	String
Type	1	Type

Table 8 shows all the real alignments that exist between the two meta-models SQL and Columnar Store.

Table 8: SQL/ Columnar - All real matches

SQL Elements	Columnar Elements
SQLElement	ColumnElement
SQLTable	ColumnStoreTable
SQLcolumn	Column
SQLcolumn	SuperColumn
String	String
Name	Name
Type	Type
Kind	Kind

4.4 NameMatching results between SQL and Document Store Metamodels

Table 9 shows the results of the application of the NameMatching heuristic on both the two meta-models SQL and Columnar Store.

Table 9. Lexical heuristic NameMatching results between SQL and Document Store

SQL Elements	Lexical NameMatching Value	Document Store Elements
Kind	0,125	Document
SQLElement	0,1	Field
SQLElement	0,1	Kind
SQLElement	0,2	Name
SQLElement	0,2	String

Kind	0,05	DocumentStoreElement
Kind	0,4	Field
Kind	1	Kind
Kind	0,3333333	String
Name	0,25	Document
Name	0,1	DocumentStoreElement
Name	1	Name
Name	0,4	Value
SQLColumn	0,2222222	Document
SQLColumn	0,2	DocumentStoreElement
SQLColumn	0,1111111	Field
SQLColumn	0,1111111	Name
SQLColumn	0,1111111	String
SQLColumn	0,2222222	Value
SQLElement	0,4	Document
SQLElement	0,4	DocumentStoreElement
SQLElement	0,2	Value
SQLTable	0,15	DocumentStoreElement
SQLTable	0,125	Field
SQLTable	0,25	Name
SQLTable	0,125	String
SQLTable	0,25	Value
String	0,125	Document
String	0,2	DocumentStoreElement
String	0,3333333	Kind
String	1	String
Type	0,125	Document
Type	0,05	DocumentStoreElement
Type	0,25	Name
Type	0,2	Value

The refinement of the results by applying the thresholds: 0.4 gives the following results (Tab.10).

Table 10: NameMatching results SQL/ Document - Threshold :0.4

SQL Elements	Lexical Matching Value	Document Store Elements
Kind	1	Kind
Name	1	Name
SQLElement	0,4	DocumentStoreElement
String	1	String

Tab.11 shows all the actual matching that exist between the two SQL and Document Store meta-models.

Table 11 : SQL/ Document Store - All real matches

SQL Elements	Document Sore Elements
SQLElement	DocumentStoreElement
SQLTable	Document
SQLcolumn	Field
String	String
Name	Name

In the next section, we will present an evaluation of the results of the application of lexical heuristics on the BDM case study, using a set of reliable mathematical measures.

5. Evaluation Based on Reliable Mathematical Measurements

We evaluated the automation aspect of our GAM approach based on four mathematical metrics: Recall, Precision, F-Measure and Overall. To define these metrics, we will present the following sets of matching:

• Set A (false negatives): contains the correct alignments, but not found automatically by our approach.

• Set B (true positives): contains the correct alignments that are found automatically.

• Set C (false positives): contains the false matching links proposed by our approach.

In the following sections we will present the quality metrics used, we consider |A| = cardinal (A), |B| = cardinal (A), |C| = cardinal (A), where cardinal represents the number of elements of the treated set.

5.1 The Recall metric

The Recall reflects the part of the actual matching among the real global matching. His mathematical formula is presented as follows:

$$Recall = \frac{|B|}{|A|+|B|}$$

This metric is between 0 and 1, its value tends to 1 if the number of matching links not detected is minimal.

5.2 The precision metric

It reflects the share of real matching among all those found, its mathematical formula is the following one:

$$Precision = \frac{|B|}{|B|+|C|}$$

This metric is between 0 and 1; its value tends to 1 if the matching errors are minimal.

5.3 The f-measure metric

It represents the harmonic average of the Precision and Recall metrics. the harmonic mean is the arithmetic mean of the inverse of the terms, defined by the following formula:

$$H = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_n}}$$

The mathematical formula of the F-Measure is therefore:

$$F\text{-}Measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

This metric is between 0 and 1, it can be considered as a global metric of quality matching calculation.

5.4 The overall metric

It quantifies the effort needed to add false negatives and remove false positives. His formula is:

$$Overall = Recall * \left(2 - \frac{1}{Precision}\right)$$

5.5 The evaluation of results

The following table 12 shows the results of the metrics found after the generation of lexical matching by the NameMatching heuristic following the BDM case study.

Table 12. Quality Measures results

	Measures			
Meta-model' Couples	Recall	Precision	F-Measure	Overall
(SQL, Key Value)	0,6	1	0,75	0,6
(SQL, Document-Store)	0,8	1	0,88888889	0,8

(SQL, Columnar)	0,875	1	0,93333333	0,875
FINAL Values	**0,85714286**	**1**	**0,92307692**	**0,85714286**

Figure 7 shows the histogram explaining the reliable values found, this diagram shows that the metrics all have values that tend towards 1 which shows the good quality of the matching found.

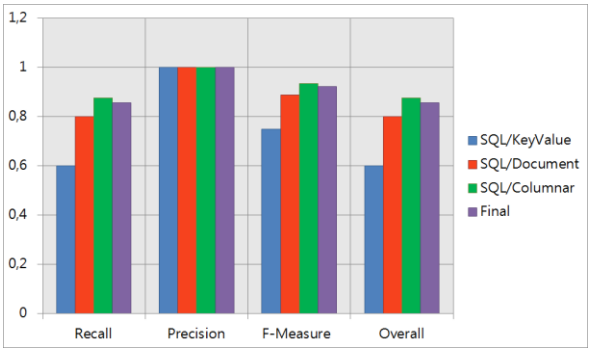

Fig.7: Measures results Histogram

6. Conclusion

In conclusion, we have presented in this article the results of our work consisting in the automatic generation of lexical matching applied on a case study composed of a heterogeneous system of SQL and NoSql databases. The final evaluation of the results found using mathematical metrics showed the validity of our heuristics.

Currently we are working on the evaluation of structural matching heuristics between heterogeneous meta-models, we are also working on the implementation of the GAM approach using .NET platform and based on M.A.S (Multi-Agents System).

References

[1] Ibn Batouta, Z., Dehbi, R., Talea, M., & Hajoui, O. Automation in code generation: Tertiary and systematic mapping review. In Information Science and Technology (CiSt), 2017 4th IEEE International Colloquium on (pp. 200-205). IEEE.

[2] Ibn Batouta, Z., Dehbi, R., Talea, M., & Hajoui, O. Multi-criteria analysis and advanced comparative study between automatic generation approaches in software engineering. Journal of Theoretical and Applied Information Technology, 2016, 81.3: 609.

[3] Ibn Batouta, Z., Dehbi, R., Talea, M., & Hajoui, O, Generative automatic matching between heterogeneous meta-model' systems Journal of Engineering and Applied Sciences 2017 (In press).

[4] Ibn Batouta, Z., Dehbi, R., Talea, ''Generative matching between heterogeneous meta-model' systems based on hybrid heuristic'' Journal of Information Technology Research, IGI Global 2018 (In Press).

[5] Morin, Brice, et al. "A generic weaver for supporting product lines." Proceedings of the 13th international workshop on Early Aspects. ACM, 2008.

[6] Schmidt, Maik, and Tilman Gloetzner. "Constructing difference tools for models using the SiDiff framework." Companion of the 30th international conference on Software engineering. ACM, 2008.

[7] Toulmé, Antoine, and I. Inc. "Presentation of EMF compare utility." Eclipse Modeling Symposium. 2006.

[8] Melnik, Sergey, Hector Garcia-Molina, and Erhard Rahm. "Similarity flooding: A versatile graph matching algorithm and its application to schema matching." Data Engineering, 2002. Proceedings. 18th International Conference on. IEEE, 2002.

[9] Lin, Yuehua, Jeff Gray, and Frédéric Jouault. "DSMDiff: a differentiation tool for domain-specific models." European Journal of Information Systems 16.4 (2007): 349-361.

[10] Xing, Zhenchang, and Eleni Stroulia. "UMLDiff: an algorithm for object-oriented design differencing." Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, 2005.

[11] Nejati, Shiva, et al. "Matching and merging of statecharts specifications." Software Engineering, 2007. ICSE 2007. 29th International Conference on. IEEE, 2007.

[12] Kolovos, Dimitrios S. "Establishing Correspondences between Models with the Epsilon Comparison Language." ECMDA-FA 9 (2009): 146-157.

[13] Hajoui, O., Dehbi, R., Talea, M., & Batouta, Z. I. An advanced comparative study of the most promising nosql and newsql databases with a multi-criteria analysis method. Journal of Theoretical and Applied Information Technology, 81(3), 579 2015.

[14] Hajoui, O ; R Dehbi ; M Talea ; Z Ibn Batouta , ''A Survey on Big Data Interoperability'', 5th International Conference on Multimedia Computing and Systems (ICMCS'16) – IEEE Conference 29 September – 1 October 2016.