International Journal of Computational Engineering Research||Vol, 04||Issue, 3||

Customized Query Interface Integration using Attribute Constraint Matrices

Sherlin Mary Koshy[1], Belfin R. V.[2]

[1] PG Student, Department of Computer Science and Engineering, Karunya University, Tamilnadu, India
[2] Assistant Professor, Department of Computer Science and Engineering, Karunya University, Tamilnadu, India

ABSTRACT:

Query Interfaces are often the only means by which certain web databases in a domain can be accessed. It has been observed that several different kinds of query interfaces exist for the same domain and they require the user to enter the same values despite the differences in the interfaces. Integrating these query interfaces is therefore essential to allow the user uniform access to all databases in a given domain. Integration of Query Interfaces using Attribute Constraint Matrices was found to be a very efficient technique for integration. This technique however performs integration over all the interfaces in a domain, while integration of only a subset of these may be necessary. This paper presents an algorithm that when applied to the result of integration of all interfaces in a domain produces an integrated attribute constraint matrix that represents only the components of those interfaces that are required to be integrated by the user.

Keywords: *Attributes, Attribute Constraint Matrix, Interface Integration, Integrated Matrix, Pruning, Query Interface, User Preference.*

I. INTRODUCTION

Online web databases contain a lot of useful information and are often accessible only by means of Query Interfaces. Query Interface is a term used to refer to the collection of HTML elements such as text boxes, radio buttons, selection lists etc., that are used to obtain information from the user. Often there are several different query interfaces in existence for a given domain requiring the user to enter the same information multiple times. For example, if a user wants to buy an air ticket online at the cheapest price he would have to visit several web sites and repeatedly enter the same information i.e., departure and arrival cities, date of journey, number of passengers etc., and then compare the price offered by these websites. Hence from the user point of view it would be easier to enter all this information in a single webpage that would appropriately access all these websites and display the results to the user in a single page. The first step to achieve this would be to generate a universal query interface that captures the structure of all the query interfaces of these websites.

There are several techniques in existence that have been summarized in [8] of which the utilization of attribute constraint matrices as defined in [7] has been found to be the most efficient. Attribute constraint matrices [7] are a far easier means of interface representation than that defined by the WISE-Integrator method [1],[2],[3] and also does not require any kind of hypothesizing of the unified interface. The integration technique suggested by [5] requires pair-wise merging which is time consuming when compared with [7]. The merging via clustering aggregation suggested in [4] will cause the attribute set to grow to be very large, while in [7] the attribute set will be restricted since a common representation is taken for the attributes of the same type. Keyword matching technique described in [6] requires additional files to store matched information, which is not required in the case of [7].

The attribute constraint matrix method in [7] first requires an ordered tree representation of the query interface based on the visual layout of the attributes. All leaf elements of the tree correspond to attributes on the query interface, a common representation for the attributes is first determined. For example in the airline domain one website may specify the terms ‘departure’ and ‘arrival’ cities while another may define them as ‘from’ and ‘to’ cities. Since they both represent the same information the ‘departure’ and ‘from’ attributes will both be represented commonly by the letter ‘a’ for example and the ‘arrival’ and ‘to’ attributes will commonly be represented by the letter ‘b’. Once these common representations have been determined the ordered attribute trees are created based on the visual layout of the attributes in the interface. From the ordered attribute trees the attribute constraint matrices are generated. These matrices are symmetric matrices having as many rows and columns as there are attributes in the interface. It has been demonstrated in [7] that these matrices can be derived from the ordered trees and vice versa. The values within the attribute constraint matrices consist of depth of the attributes and the distance between the attributes derived from the corresponding ordered tree. These matrices are then suitably expanded and merged as per the operations suggested in [7] to yield the final unified attribute constraint matrix. From this unified attribute constraint matrix the ordered tree representation of the final unified interface can be easily determined.

The performance of this technique [7] was measured in terms of the extent to which the structure of the existing interfaces was maintained in the integrated interface i.e., the extent to which the constraint imposed by each interface was satisfied in the unified interface. There were three constraints described in [7] hierarchical, group and precedence and the satisfaction rate of these constraints were found to be acceptable. Hence we can conclude that the technique of using attribute constraint matrices for generating the unified query interface for all the interfaces in a domain is highly efficient and easy to implement.

It may often be necessary to integrate only a sub set of the interfaces in a domain, for example if only the results of query interfaces within a region is needed, i.e., a user wants to compare the air ticket prices offered by only the travel operators in his country of residence. Or if user wants to view results from websites that offers certain privileges. There are two ways to approach this problem as defined in [9], one is the bottom up approach which involves performing integration from scratch on only the interfaces that are necessary, and the other is the top down approach which involves reducing a global integrated tree representation to contain only the attributes of the interfaces that are preferred. This top down approach called pruning [9] is defined to work on the global integrated tree. The aim of this paper is to perform the same pruning operation on the global unified attribute constraint matrix generated by [7].

Pruning the integrated tree is a far more efficient technique than generating the integrated tree bottom up because the pruning operations become easier with an increase in the number of the interfaces that need to remain. This is because with an increase in the number of interfaces that need to remain there will be fewer attributes that need to be removed from the integrated tree hence fewer operations that need to be performed, this is not the case with the bottom up approach since more attributes will need to included in the integrating process resulting in an increased amount of time being taken.

The rest of this paper is organized as follows. In section 2, the top down tree pruning technique described in [9] is explained. This serves as the basis of the main work in this paper. In section 3, an algorithm is presented and explained that performs the same pruning operations except that the algorithm takes as input the unified attribute constraint matrix generated by [7] as the input. In section 4, the results after implementing this method are given. Finally, in section 5, the conclusions are described.

II. TOP DOWN TREE PRUNING

This technique described in [9] works on the integrated tree schema that has been generated using the methods described in [1] and [5]. The essential step is to remove from the integrated schema those fields that are non-existent in the interfaces that are required. Once fields are removed there needs to be further operations that need to be done to refine the final tree.

The suggested technique has three primary steps, Remove, Collapse and Split-Up. The remove step involves removing from the integrated tree all the leaf nodes that do not exist in the interfaces being considered, this step also recursively removes all ‘fake’ leaves, i.e., those internal nodes that become leaves in the process of removal. The send step collapses a child node with its parent if it is the only child, this step is also performed recursively. The final step splits up the nodes in the tree on the basis of the groups in the required schemas if they share the same parent. However it has been stated in [9] that the split up operation need not be performed since the original group was already validated with respect to the original integrated interface. Hence this step is not considered for the development of the algorithm, in addition the split operation will result in violating the integrity of the unified interface. Figure. 1 depicts the two steps remove and collapse given an initial integrated tree.

Figure 1(a) shows an integrated tree with five leaf nodes ‘a’, ’b’, ’c’, ’d’ and ’e’. Assuming that the interfaces that are required contain only the attributes ‘a’, ‘b’ and ‘d’, the leaf nodes ‘c’ and ‘e’ will then have to be removed. Figure 1 (b) shows integrated tree with the leaf nodes ‘c’ and ‘e’ removed, this is the removal step. Since ‘e’ was a child of the root it’s removal causes no problem, however, removing the leaf node ‘c’ resulted in its sibling ‘d’ becoming an only child of its parent. Hence node‘d’ is collapsed with its parent to become a child of the root. This is the collapse operation, depicted in Figure 1(c). On completion of the collapse operation the integrated tree obtained will have only the attributes of those interfaces that are required.

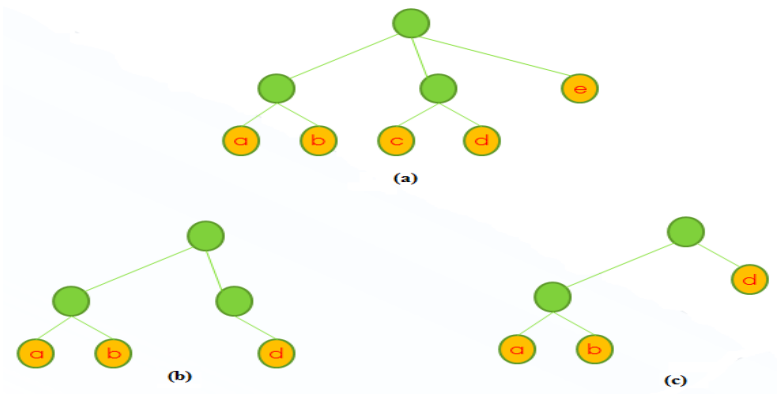

Figure 1. Pruning the Integrated Tree

III. PRUNING ALGORITHM FOR INTEGRATED ATTRIBUTE CONSTRAINT MATRIX

The pruning algorithm for unified constraint matrices is given in Figure 2, the unified attribute constraint matrix 'M' generated using the technique in [7] as well as the set 'Q' of all the interfaces that are required by the user to be integrated is given as the input. The output of this algorithm will be the matrix 'QI' containing only the attributes of those interfaces required by the user. The ordered tree for this matrix QI can be easily generated using the technique described in [7]. The algorithm is followed by an example of its operation on attribute constraint matrices based on the integrated tree in Figure1.

Algorithm **PruneUnifiedMatrix** (M, Q)

Input:
M: Unified attribute constraint matrix of all interfaces in domain
Q: Set of query interfaces specified by user.

Output:
QI: Integrated attribute constraint matrix of user specified matrices

1. QI=M
2. Generate set SET1 of all attributes in QI.
3. Generate set SET2 of all distinct attributes in Q.
4. Determine groups of siblings Sib1, Sib2, Sib3.....Sibn in QI.
5. SETa=SET1-SET2.
6. Remove from QI all attributes in SETa.
7. **If** any attribute or combinations of attributes 'att' in SETa ϵ{Sib1~Sibn}

```
A= {Sib_i - att}
If |A|=1                          // lone attribute after siblings are removed
        b=QI [A,A]                // current depth of lone attribute
        QI [A, A] =1              // set depth of lone attribute in QI to 1

        for (i=0;i<sizeofQI;i++)
        QI [i, A] =QI [i, A]-(b-1)   // update distance of all other attributes to lone attribute

End If
```

Figure 2. Pseudo code of algorithm PruneUnifiedMatrix

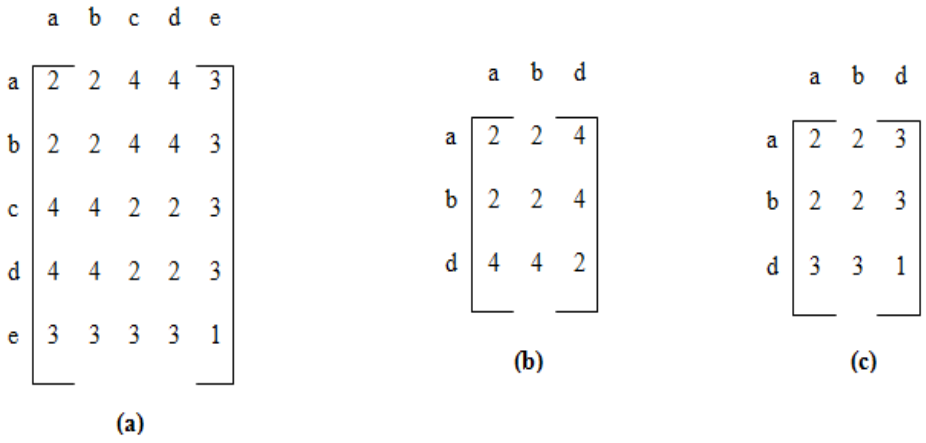

Figure 3. Attribute constraint matrix values at different stages in the algorithm PruneUnifiedMatrix

Figure 3(a) shows the attribute constraint matrix corresponding to the tree in Figure 1(a), matrix is generated using the technique described in [7] with the diagonal values representing the depth of the node from the root, and the remaining values representing the number of edges to be travelled between the corresponding nodes. Figure 3(b) shows the matrix after the remove operation in line 6 of the algorithm in Figure 2 has been executed, the rows and columns corresponding to the attributes 'c' and 'e' have been removed which are assumed to be those attributes not present in the interfaces required by the user to be integrated. Figure 3(c) shows the final attribute constraint matrix after the entire algorithm in Figure 2 has been executed. After the remove operation the attribute 'd' was found to be the lone attribute since it's sibling 'c' was removed, it's depth has been set to 1 and it's distance to the remaining attributes has been reduced by the extent to which it's depth has to reduce to become 1. Notice that the matrix in Figure 3(c) will be the same as the matrix drawn from Figure 1(c) using the technique in [7], hence showing that this algorithm works correctly on the attribute constraint matrices suggested by [7].

IV. RESULT ANALYSIS

The proposed algorithm in Figure 2 was implemented in the Java language with the input as the examples given in [7], the satisfaction rate of hierarchical and precedence constraints defined in [7] was satisfactory, however the satisfaction rate of the group constraints [7] was found to reduce by a small extent when compared to the group constraint satisfaction rate of the initial integrated interface, this is due to the existence of lone siblings after the removal process. Overall this algorithm for customizing the integrated query interface as per the users' preference does not violate the constraints on the structures of the constituent schemas

V. CONCLUSIONS

Large scale integration of query interfaces is a widely studied problem in deep web data integration. Representation of the query interfaces as attribute constraint matrices was found to be capable of capturing all the constraints imposed on a query interface. The final generated unified matrix facilitated the development of a unified interface that maintained all the characteristics of the constituent matrices. Customizing this integration according to user preference involved developing an algorithm that could be applied on this unified matrix such that it will ultimately contain only the attributes of those interfaces specified by the user. Java programming language was used to implement the core algorithm and a unified matrix was generated from the given input matrices using the proposed algorithm which did not violate the constraints imposed by the constituent schemas.

REFERENCES

[1.] [1] H. He, W. Meng, C. Yu and Z. Wu, *WISE-Integrator: an automatic integrator of web search interfaces for e-commerce*, VLDB'03 (2003).

[2.] [2] H. He, W. Meng, C. Yu and Z. Wu, *Automatic integration of web search interfaces with WISE-Integrator*, The VLDB Journal 13 (2004) 256–273.

[3.] [3] H. He, W. Meng, C. Yu and Z. Wu, *WISE-Integrator: a system for extracting and integrating complex web search interfaces of the Deep Web*, VLDB'05 (2005) 1314–1317.

[4.] [4] W. Wu, C. Yu and A. Doan, *Merging Interface schemas on the Deep Web via clustering aggregation*,ICDM'05 (2005).

[5.] [5] E. Dragut, W. Wu, P. Sistla, C. Yu and W. Meng, *Merging source query interfaces on web databases*, ICDE'06 (2006) 1–10.

[6.] [6] F. Yuan, L. Han and Y. Wei, *A Deep Web interface integration approach based on keyword matching and similarity computing*, Journal of Computational Information Systems 6 (2009) 1569–1576.

[7.] [7] Y. Li, Y. Wang, P. Jiang and Z. Zhang, *Multi-Objective Optimization Integration of Query Interfaces for the Deep Web based on Attribute Constraints*, Data & Knowledge Engineering 86 (2013) 38-60

[8.] [8] Sherlin et al., *A Study on the Existing Techniques for Query Interface Integration*, International Journal of Advanced Research in Computer Science and Software Engineering 3(12), December - 2013, pp. 330-333

[9.] [9]E.C. Dragut, F. Fang, C. Yu, W. Meng, *Deriving customized integrated web query interfaces*, IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Workshops, 2009, pp. 685–688.