Clustering using Fast Structural Hierarchy Extraction from User-Defined Tags for Semi-Structural Languages

Hyun-Joo Moon, Haeng-kon Kim, Eun- Ser Lee, *Member, IAENG*

Abstract—Needs for semi-structured Languages that use user-defined tags like XML or other Markup languages are extremely growing because of the explosive increase of Internet usage, heterogeneous computing environments, and ubiquitous computing technologies. Tree traversing schemes for syntactic and semantic comparison of user-defined tags is one of the key issues for success on the clustering, and Web data clustering is important for management of huge amounts of Web-based data. However, excessive tree traversing causes expensive time and space consumption. With the rapid growth of Web data usage and the importance of the management, comparison techniques such as similarity detection without tree traversing are more and more needed for efficient information and database management. This paper introduces a fast structural hierarchy extraction technique without tree traversing on schema parse trees on semi-structured language format data. This technique uses Direct Invariant Encoding Scheme and manipulates tag sequences same order as Depth First Search tree traversing method. We use XML schema, DTD, extracted from XML data to evaluate our fast structural hierarchy extraction technique. We also use ontological similarity comparison technologies, and those are based on WordNet. For extraction of structural hierarchy, we apply LNS (Longest Nesting common String) extraction method. With this technique, semi-structured Web data management can be easier and faster than any existing tree traversing methods.

***Index Terms*—Semi-Structured Languages, XML DTD, Similarity Detection, User-defined tags**

I. Introduction

Semi-structured languages such as XML [1] need time and efforts because of the use of their user-defined tags. As the usage of semi-structured Web data increases, the need for clustering and managing Web data also increases. However, the comparison of semi-structured language format Web data has difficulties because of the semi-structured characteristics, in where user-defined tags cause semantic ambiguity, different tag names may have same meanings on semantics, or same tag names may have different meanings in different structures. Some of these ambiguity problems in semi-structured languages can be solved by ontological techniques, and some can be solved by both syntactic and semantic structural analysis. Our free-traversing technique uses WordNet as its ontological solution for semantic comparison, and also uses multi-tag sequences methods as its time solution for syntactic comparison. This paper consists of 5 sections. Section 2 summaries related works, and Section 3 describes our system architectural layout and free-traversing technique. In Section 4, we evaluate our technique by some experiments. Finally, we conclude the result in Section 5.

Hyun-Joo Moon is an adjunct professor of the Department of Cultural Contents, Hankuk University of Foreign Studies (HUFS), Seoul, 130-791, Korea (phone: +82-11-9151-0335; fax: +82-2-824-3862; e-mail: hyunjoomoon@gma il.com).

Eun-Ser Lee is a professor of the Department of Computer Engineering, Andong National University, Gyeongsangbuk-do, 760-749, Korea (e-mail: eslee@andong.ac.kr).

Haeng-kon Kim is a professor of the Department of Computer Information & Communication Engineering, Catholic University of Daegu, Gyeongsangbuk-do, 712-702, Korea (e-mail: hangkon@cu.ac.kr).

II. Related Works

Related to the similarity detection algorithm, there are some categories. One of the initial comparison methods is Tree Edit Distance Algorithms. These algorithms can detect changes in structured documents or semi-structured documents and fix one of them with its graph edit operations. Valiente's algorithm, Tai's tree-to-tree correction algorithm, and Shasha and Zhang's algorithm are all based on atomic edit operations on nodes, and can not consider subtree level operations [4][8]. The second type is Structure-based Document Differencing Algorithms. LaDiff and MH-Diff included in the type use edit scripts based on the weighted matching. These algorithms do not make good results in semi-structured documents or in documents having node duplications [2][5][7]. The third is Time Sequence-based Algorithms [9] such as Time Warping methods or DFT-based methods, deal with document comparison as frequencies. They are totally different from other methods based on graph matching or tree traversing algorithms that are generally computationally expensive. These algorithms focus on the computation of structural similarity for fast and effective results of the management of Web data. They provide a significant reduction of the required computation costs. However, they do not have any solution for semantic problem from ambiguity of semi-structured languages. The last one is XClust [10] providing ontological semantic solution. XClust uses a tree traverse algorithm including PCC (Path Context Coefficient) concepts to capture the degree of similarity in the paths of two elements. The ontological solution and tree traversing causes a serious processing time delay, especially in large-scale databases or dissimilar document groups. That is, XClust is not a practical solution for semi-structured Web data.

III. Structural Hierarchy Extraction

A. System Architecture Layout

Our structural hierarchy extraction system, named XJune [12, 13], is implemented to compare semi-structured Web data without parse tree traversing. Therefore, we use DIES (Direct Invariant Encoding Scheme) encoding method and it generates same results as DFS (Depth First Search) of parse tree traversing without tree traversing.

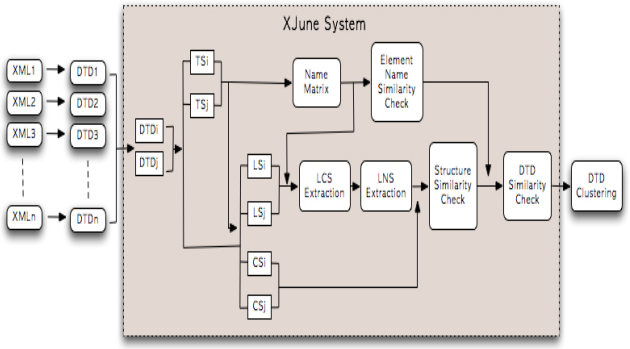

Figure 1. Structural Hierarchy Extraction System

Figure 2 shows DIES encoding scheme used on XJune. Each tag consists of a start tag and an end tag.

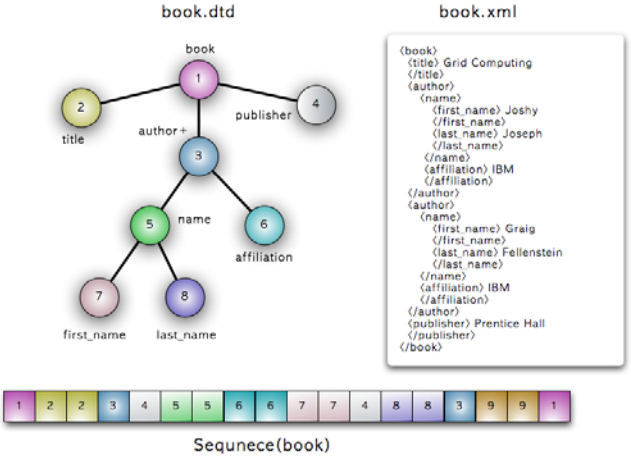

Figure 2. DIES Encoding Scheme

B. User-defined Tag Comparison

User-defined tag comparison including ontological techniques needs very expensive time consumption. However it can be a solution for ambiguity caused by user-defined tags of semi-structured languages. XJune sends each node name of trees to WordNet ontology system and uses our additional functions to address this problem. The similarity values between two tags are from 0 to 1. 0 means they don't have any similarity, and 1 means they are exactly same. We limit the steps calls of WordNet less than 3. Figure 3 describes the ontological comparison between trees as a graphical view and a table view.

	author	name	first_name	last_name	affiliation	email
book	0.0	0.0	0.0	0.0	0.0	0.0
title	0.0	0.0	0.0	0.0	0.0	0.0
writer+	0.81	0.0	0.0	0.0	0.0	0.0
name	0.0	1.0	0.0	0.0	0.0	0.0
given_name	0.0	0.0	0.81	0.0	0.0	0.0
family_name	0.0	0.0	0.0	0.81	0.0	0.0
affiliation	0.0	0.0	0.0	0.0	1.0	0.0
publishing_company	0.0	0.0	0.0	0.0	0.0	0.0

Figure 3. Matrix for User-defined Tag Comparison

`TagNameSim` algorithm in Figure 4 shows the algorithm for the comparison of two XML DTDs. For this, `TagSim`, the values in the table in Figure 2, should be calculated. We assume that each tag is an element. Also, we will use tree structure for easy understanding even though we use encoded multi-tag sequences instead of parse tree. An element in DTD1 is $e_{1i} \in d_1$ and an element in DTD2 is $e_{2j} \in d_2$, In here, $1 \le i \le n$ and $1 \le j \le m$. The node number of DTD1, n, is defined as $\mathtt{NodeNumber}(d_1)$, and m is defined as $\mathtt{NodeNumber}(d_2)$. Tag name comparison for user-defined tags is shown in Figure 3.

TagSim(t_1, t_2, threshold$_1$, threshold$_2$, map)

Input: t_1, t_2 - two DTD trees to be compared with each other
threshold$_1$- The minimum ElementNameSim value to have the same tag index
threshold$_2$- The minimum ElementNameSim value to be included in LocalMatch
map - the map that links each tag name to an index Number
Output: the degree of name similarity between the two DTD trees

```
Begin
  a1 ← TreeTraverse(t1)
  a2 ← TreeTraverse(t2)
  next ← 0
  matrix ← {}
  for each e1∈a1 do
    for each e2∈a2 do
      n1 ← e1.name
      n2 ← e2.name
      n = TagSim(n1, n2, 3)
   if n > threshold1 then
    if both names are not exists in map then
      put (n1, next) into map
        put (n2, next) into map
    next ← next + 1
    else if n1 is not exist in map then put (n1, index of n2) into map
     else if n2 is not exist in map then put (n2, index of n1) into map
  else
      if n1 is not exist in map then
        put (n1, next) into map
        next ← next + 1
     if n2 is not exist in map then
        put (n2, next) into map
        next ← next + 1
    matrix ← matrix ∪ (e1, e2, n )
    MatchList ← LocalMatch(matrix, |a1|, |a2|, threshold2)
    return ∑sim / Max(|a1|, |a2|)
      sim ∈ MatchList
End.
Function TreeTraverse(t)
Input:   t - a DTD tree
Output: an element array using visitor pattern which traverse tree
        in depth-first mechanism
 TagNameSim(w1, w2, maxDepth) = OntologySim(w1,
w2, maxDepth)
```

Figure 4. User-defined Tag Comparison

C. Structural Hierarchy Extraction

For structural hierarchy, we use LCS (Longest Common String) and LNS (Longest Nesting common String). LCS is the same tags, which means the common nesting elements on

two DTDs. LCS occasionally includes some broken tags generated by an end tag and a different start tag that have the same encode. The problem occurs from the encoding scheme using the same ending number for a start tag and its end tag. To solve this problem we use LNS [12, 13] to filter the broken tag pairs. Figure 5 shows the LCS and LNS extraction process but in this case LCS is exactly same as LNS, which means that there are no broken tags.

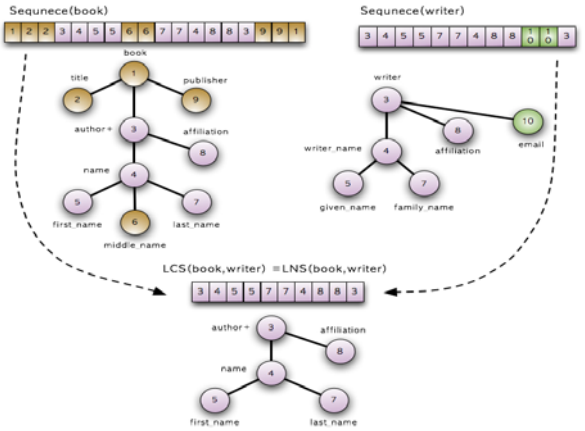

Figure 5. Structural Hierarchy Extraction Flow

By using each level of tags and their parent tag information, LNS can detect broken tags. With LCS and LNS, we successfully extract the same structural similarity from semi-structured user-defined tags. Using ontological techniques from WordNet and XJune, we could extract the tag name similarity, and now we can extract the structural hierarchy similarity. Using this syntactic and semantic similarity detection method allows us to generate clear and precise comparison of Web data. The whole structural hierarchy comparison algorithm is shown in Figure 6. This algorithm uses LCS and LNS from multi-level tag sequences to compare two Web data.

Algorithm StructuralHierarchyExtraction(t_1, t_2, map)

Input: t_1, t_2 – two XML DTD trees
map tag name to index map
Output: structural hierarchy similarity between two DTD trees

Begin
length ← 0
x_1 ← 0
x_2 ← 0
seq_1 ← TagSequence(t_1)
seq_2 ← TagSequence(t_2)
for (match ← NextBestMatch(x_1, x_2)) ≠ nil do
length ← match.length
x_1 ← match.i + match.length
x_2 ← match.j + match.length
return length / ((|seq_1|+ |seq_2|)- length)
End.
Function TagSequence(t) using visitor pattern
Function NextBestMatch(seq_1, seq_2, m_1, m_2)
Input:
seq_1, seq_2: tag sequences for each DTD
m_1, m_2: start index
Output:
Next maximum-length match
if there's no more match then nil
Begin
if m_1 >= |seq_1| or m_2 >= |seq_2| then return nil
if $seq_1[m_1] = seq_2[m_2]$ then return (m_1, m_2, Length(seq_1, seq_2, m_1, m_2))
else
a ← Search(seq_1, seq_2, m_1, m_2)
b ← Search(seq_2, seq_1, m_2, m_1)
if a = nil and b = nil then return nil
else if a = nil then return (b.j, b.i, b.length)
else if b = nil then return (a.i, a.j, a.length)
else if a.length >= b.length then return (a.i, a.j, a.length)
else return (b.j, b.i, b.length)
End.
Function Search(seq_1, seq_2, m_1, m_2)
Begin
for each $p_{m1<=i<|seq1} \in seq_1$ do
for each $q_{m2<=j<|seq2|} \in seq_2$ do
if $p_i = q_j$then return (i, j, Length(seq_1, seq_2, i, j))
return nil
End.
Function Length(seq_1, seq_2, m_1, m_2)
Input:
Output:
The length matched from corresponding offset
in each sequence

Figure 6. Structural Hierarchy Extraction

D. Comparison of Semi-structured Web Data using Structural Hierarchy Extraction

In Figure 7, we can see the algorithm of the comparison of two XML schema DTDs using structural hierarchy extraction. The calculated comparison results are added with their weights. We set the default weights of α and β as 0.4 and 0.6 for the simulation.

Algorithm SemiStructureComparison(t_1, t_2)

Input:
t_1, t_2 - two XML DTD tree
Output:
DTD similarity

Begin
initialize map <as name to index>
return α * TagSim(t_1, t_2, map)
+ β* StructuralHierarchyExtraction(t_1, t_2, map)
End.

Figure 7. Comparison for Semi-Structured Web Data

IV. EXPERIMENTS

A. Experimental Environment

XJune system is implemented by Java SE 6.0 on Windows XP SP2 on Pentium M1.6 GHz RAM 1.56GB. We use JaxMe 2 DTD Parser to parse XML schema DTDs. XJune is running with a user interface in Figure 8 for easy manipulation of the thresholds and weights for XML schema DTD comparison.

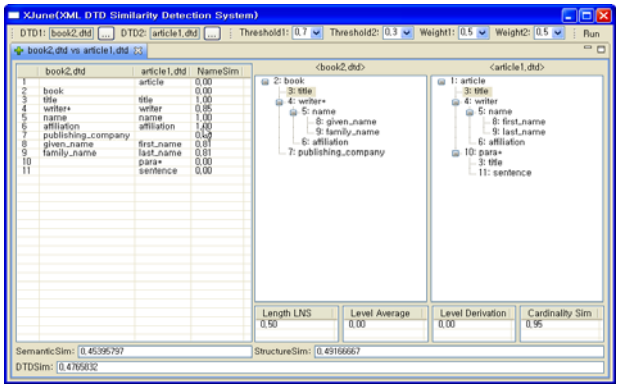

Figure 8. User Interface for Web Data Comparison

B. Experimental Results

For the experiments, we use 4 DTDs in Figure 9; Book1, Book2, Author1, and Article1. These four DTDs have some similarities and some differences. For example, Book1 and Book2 are almost same except some user-defined tag names which have same meaning.

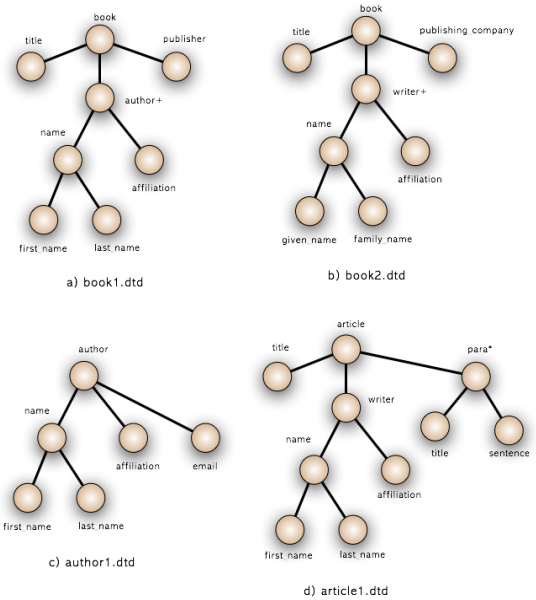

Figure 9. Sample DTDs for the Experiments

There are 6 possible combinations for the comparison, and we show 3 cases of them in this paper. Figures 10, 11, and 12 show the results.

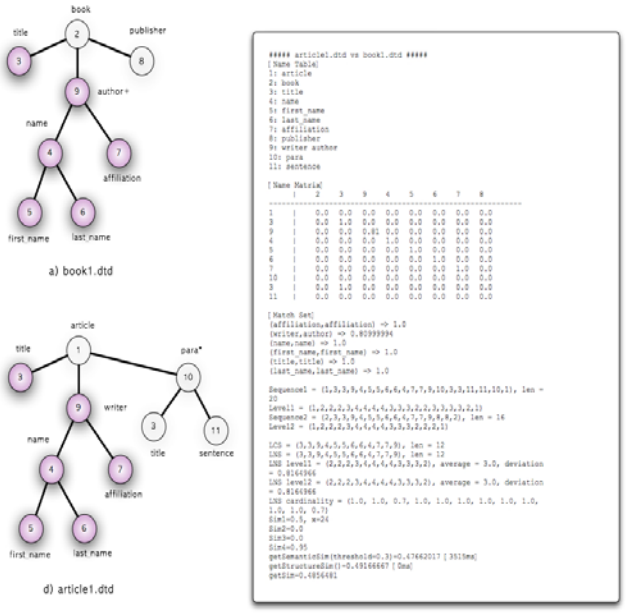

Figure 10. Comparison Result on Book1 and Article1

XClust system introduced ontological similarity detection mechanism for semi-structured language XML. We use the same DTDs on XClust system, and the results are shown in Figure 13. XClust traverses the 2 DTD trees N+2 times for each comparison for two elements in each DTD, when we assume that the number of children of the element is N.

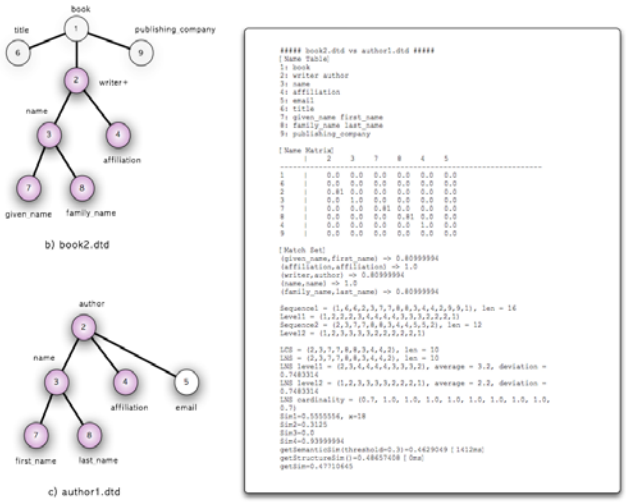

Figure 11. Comparison Result on Book2 and Author1

XClust has a good idea for extracting the semantic similarity comparison, but the system has an important disadvantage in an aspect of cost and time.

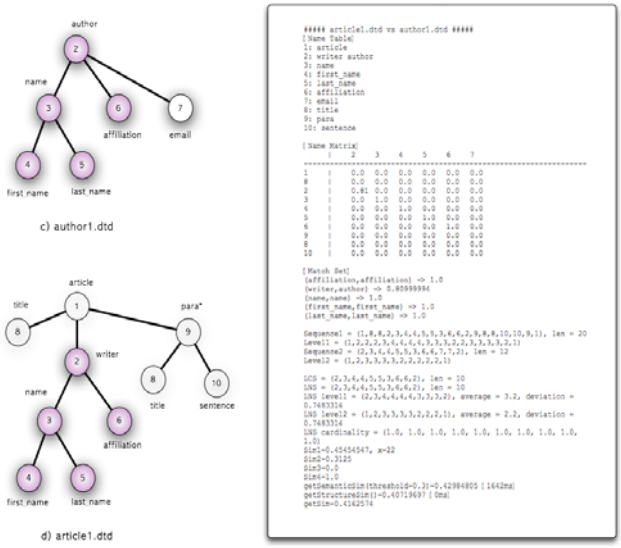

Figure 12. Comparison Result on Author1 and Article1

In the results, the processing time consumption is extremely expensive. That is, extremely low speed when we compare the results with XJune system. XJune uses additional ontological techniques, but comparison speed is very practical. When the numbers of nodes on DTD trees are increased, XJune can have the fast processing speed but XClust has serious processing time delay because of tree traversing numbers. These experimental results show that LNS extraction is one of the solutions for Web data processing for the future.

We tested more than 200 DTDs for the evaluation of XJune. As we have already mentioned, XClust is an excellent approach to compute similarity using ontological techniques, but it has problems such as low processing speed and dependency on the number of elements or traversing. They cause lack of generality and practicality. For example, if one of the DTDs is a subset of the other, it should be different from the case that two DTDs are different but have some same elements. XClust also has a problem when it compares one small document with one big document. The system has a great comparison result when it compares similar sized

documents, but it is not good at comparison for totally different sized documents.

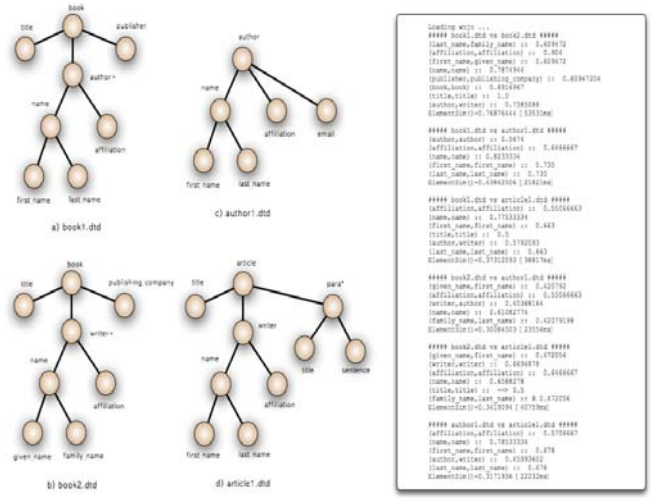

Figure 13. Comparison Results on XClust

We know that the management of huge amount of Web data is very important for the future, and semi-structured Web data, such as XML, will be more and more increased. With some metadata such as multi tag sequences of element level average and deviation, semi-structured Web data can be managed easily. Existing structural similarity detection methods cannot deal with semantic similarities, especially for semi-structured languages. However, free-traversing technique can address these problems as you can see in Figure 14.

	Tree-distance	Structure-based	XClust	XJune
Analysis Method	Structure	Structure	Structure &Semantic	Structure &Semantic
Ontology	N/A	N/A	Available	Available
Additional Ontology	N/A	N/A	N/A	Available
Processing Speed	Medium	Fast	Extremely Slow	Fast
1:M Mapping	N/A	N/A	Available	Available
# of Tree Traversing	N times /each comp.	None	N+2 times /each comp.	None

Figure 14. Evaluation of Algorithms

V. CONCLUSIONS

User-defined tags are used in many markup languages and have lots of advantages such as flexibility and scalability. However they cause lots of ambiguities which are hard to manipulate in way what we originally intend. Managing semi-structured format data is one of the time-consuming tasks and needs lots of time and effort. Structural hierarchy extraction technique is useful for comparison for various fields of semi-structured language format data processing. Heterogeneous and dynamic environment and increase of Internet usage make a need for fast and effective techniques for the management of huge amount of Web data, and additionally XML-based Web data has a serious problem of ambiguity from its semi-structured user-defined tag structures. The problem costs expensive time and space consumption, and the cost is getting much growing more and more with the explosive Web usage.

This paper proposes a structural hierarchy extraction technique using LNS extraction scheme for a solution dealing with semi-structured Web data. Using this comparison technique gives many advantages on both flexible semantic compatibility and practical processing time. When we consider further extension of Internet and Web usages, this technique can be one of good solutions for the future management of semi-structured Web data. This comparison technique provides fast processing time and reasonable cost for dealing with semi-structured data for the future.

REFERENCES

[1] Extensible Markup Language (XML) 1.0, Available on http://www.w3c.org/TR/REC-xml, (1998)

[2] Andrzej Goscinski, Mirion Brearman, “Resource Management in Large Distributed Systems”, *ACM SIGOPS, Operating Systems Review Vol. 24*, 1990, pp. 7–25.

[3] W. Lian, D. W. Cheung, S. Yiu “An Efficient and Scalable Algorithm for Clustering XML Documents by Structure”, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 1. Jan., (2004)

[4] G. Costa, G. Manco, R. Ortale, A. Tagarelli “A Tree- Based Approach to Clustering XML Documents by Structure”, PKDD'04, LNAI 3202 (2004) 137-148

[5] P. Klein, S. Tirthapura, D. Sharvit, B. Kimia “A Tree-edit -distance Algorithm for Comparing Simple, Closed Shapes”, Proceedings of the 11th Annual ACM SIAM Symposium of Discrete Algorithms (2000) 696-704

[6] T. Dalamagas, T. Cheng, K. J. Winkel, T. Sellis “Clustering XML Documents Using Structural Summaries”, EDBT'04 Workshops, LNCS 3268 (2004) 547-556

[7] E. Borenstein, E. Sharon, S. Ullman “Combining Top-down and Bottom-up Segmentation”, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Jun., (2004)

[8] R. A. Ekram, A. Adma, O. Baysal “diffX: An Algorithm to Detect Changes in Multi-Version XML Documents”, Proceedings of the 2005 Conference on the Centre for Advanced Studies on Collaborative Research (2005)

[9] K. Zhang, J. T. Wang, D. Shasha “On the Editing Distance between Undirected Acyclic Grahphs and Related Problems”, Proceedings of the 6th Annual Symposium of Combinatorial Pattern Matching (1995)

[10] D. Rafiei, A. Mendelzon “Similarity-Based Queries for Time Series Data”, Proceedings of the ACM International Conference on Management of Data, May, (1997) 13-24

[11] M. L. Lee, L. H. Yang, W. Hsu, X. Yang “XClust: Clustering XML Schemas for Effective Integration”, Proceedings of the 11th ACM International Conference on Information and Knowledge Management, (2002) 292-299

[12] H. J. Moon, K. J. Kim, G. C. Park, C. W. Yoo, “Effective Similarity Discovery from Semi-structured Documents”, International Journal of Multimedia and Ubiquitous Engineering, Vol. 1, No. 4 (2006) pp12-18

[13] H. J. Moon, S. H. Kim, J. B. Moon, E. S. Lee, "An Effective Data Processing Method for Fast Clustering", ICCSA'08, (2008)