DOI: 10.3745/KIPSTD.2010.17D.5.353

무선 환경에서 XML 조각 스트림 질의 처리를 위한 효율적인 레이블링 기법

고 혜 경†

요 약

전통적인 데이터베이스와 달리, XML 스트림에 대한 질의는 실시간 처리와 메모리 사용량에 제한이 있다. 이 논문에서는 XML 조각들 사이의 구조적인 관계를 빠르게 확인할 수 있는 강력한 레이블링 기법을 제안한다. 제안된 레이블링 기법은 많은 중복된 작업과 처리해야 하는 조각들의 수를 최소화하여 효율적인 질의 처리를 제공한다. 실험 결과, 제안된 레이블링 기법은 효율적으로 질의를 처리하고 메모리 사용량을 최소화 할 수 있다.

키워드: XML 조각 스트림, 질의 처리, 레이블링, 무선 환경

Efficient Labeling Scheme for Query Processing over XML Fragment Stream in Wireless Computing

Hye-Kyeong Ko†

ABSTRACT

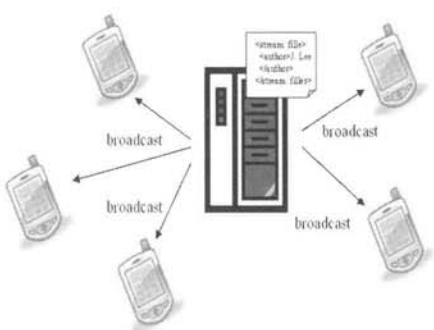
Unlike the traditional databases, queries on XML streams are restricted to a real time processing and memory usage. In this paper, a robust labeling scheme is proposed, which quickly identifies structural relationship between XML fragments. The proposed labeling scheme provides an effective query processing by removing many redundant operations and minimizing the number of fragments being processed. In experimental results, the proposed labeling scheme efficiently processes query processing and optimizes memory usage.

Keywords: XML Fragment Stream, Query Processing, Labeling, Wireless Computing

1. Introduction

XML [11] is emerging as a defacto standard for information representation and data exchange over the web. XML data, being inherently hierarchical and semi-structured, poses a heavy overhead on runtime factors, such as memory requirement and processing efficiency [2]. With the increase of mobile devices and with the request for information on the move, a server disseminates data over low-bandwidth and error-prone environments [2]. As the intermittent connectivity of mobile clients makes infeasible to carry large dataset, fragments may be a better choice for data transmit. This paper focuses on an efficient query processing over continuous XML fragment streams. The server disseminates XML data to multiple clients concurrently through a data stream, while each client is completely responsible for processing the stream.

(Fig. 1) shows an example of disseminationon of XML



(Fig. 1) Dissemination of XML fragments

† 정 회 원 : 한국과학기술원 전산학과 연구교수
논문접수 : 2010년 2월 24일
심사완료 : 2010년 4월 5일

fragments. If queries are processed on large XML document, XML fragment should demand less memory and processing power [2], [4]. Recently, many research works focus on answering queries on streamed XML data which deal with fragmented XML data based on Hole-Filler model [4], such as XFrag [2], XFPro [6].

However, Hole-Filler model has two main limitations:

The space overhead for the hole/filler IDs with related XML tag in XML fragment could be very large.

Given two XML fragments, it can not directly identify the ancestor-descendant (A-D) relationships between fragments.

The principal problem with Hole-Filler is the limited effectiveness in reconstructing portions of the original XML document in the limited memory and evaluating queries against XML fragment. Due to these limitations, the required memory for query processing over streamed XML fragments would be increased. In order to solve this problem, the location information of fragments would be known. For this purpose, the labels of fragments can be utilized, and the relationship of fragments can thus be obtained through these labels.

A number of labeling schemes have been designed to label nodes, namely region-based labeling [16] and prefix labeling scheme [15]. However, when using the region-based labeling scheme for streamed XML fragment model, identifying the parent-child relationship of fragments is difficult, because only the region of the node label is represented. In the prefix labeling scheme [15], the nodes inherit their parent's labels as the prefix to their own labels. If XML tree is deep, the label size will be increased with depth. Thus, query processing cost will be high in the two labeling schemes over streamed XML fragment. Consequently, a new labeling scheme is strongly required for identifying portions of the original XML document against XML fragment.

In this paper, a novel labeling scheme for simply identifying structural relationships between XML fragments by borrowing Hole-Filler model. The contributions of this paper include the following:

A robust labeling scheme is proposed, which can quickly identify structural relationships between XML fragments.

The experimental results demonstrate the effectiveness of the proposed scheme.

The paper is organized as follows. Section 2 presents the related work of XML stream query processing. Section 3 presents the proposed labeling scheme for XML fragment stream. Section 4 shows the conducted experimental results and Section 5 provides our conclusion.

2. Related Work

Several recent works have focused on frameworks for continuous processing of data streams [3], [5], [8], [9], [17], [18]. There is, however, no work done in stream query processing of fragmented XML data. Hole-Filler model was first proposed in [7]. However, it is used in the context of pull-based content navigation over mediated views of XML data from disparate data sources. Recently, many research works address query processing on streamed XML fragments based on Hole-Filler model, such as XFrag [2], XFPro [6]. In XFrag [2], XML fragments are processed when they arrive and only these fragments that may effect on the query results are kept in the association table. However, XFrag pipeline is still space consuming in maintaining the links in the association tables and time cost in scheduling the operations for each fragment [6]. Moreover, the "redundant" operations are caused with dependence between adjacent operators in XFrag. XFPro [6] presents a framework and a set of techniques for processing XPath queries over streamed XML fragments and techniques for enabling the transformation from XPath expression to optimized query plan. In XFPro, the query processing pipeline is improved over XFrag because it just considers subroot nodes. However, many intermediate fillers occur for queries including "*" or "//".

3. Efficient Labeling Scheme on Hole Filler Model

3.1 Preliminary of Hole-Filler Model

XML document is pruned in to many fragments in Hole-Filler model [7]. In order to summarize the structure of XML fragments, *tag structure* is used to support structural and fragment information. A *tag structure* is itself structurally a valid XML fragment that fulfills to XML schema or DTD, where a *tag* corresponds to an XML element and is qualified by a unique *id*, and the element tag name [7].

(Fig. 2) shows an XML document, which acts as a running example of our work. Given an XML document tree T_d = (V_d, E_d, Σ_d, $root_d$, Did), a filler T_f = (V_f, E_f, Σ_f, $root_f$, fid, $tsid$) is a subtree of XML document associating an fid and a $tsid$, where V_f, E_f, Σ_f is the subset of node set V_d, edge set E_d and element type set

```
<book>
        <title> XML </title>
        <authors>
            <author> J. Lee </author>
            <author> H. Park </author>
        </authors>
        <year> 2007 </year>
        <section>
            <head> XML DTD </head>
            <subsection> … </subsection>
        </section>
            …
</book>
```

(Fig. 2) Example of XML document

Σ_d respectively, and $root_f$ ($\in V_f$) if the root element of the subtree; a hole H is an empty $node_v$ ($\in V_d$) assigned with a unique *hid* and a *tsid*, into which a filler with the same *fid* could be positioned to complete the tree [7].

(Fig. 3) depicts the *tag structure*. Given an XML document tree, we can fragment it by recursively inserting a hole at every point where a subtree is pruned, i.e., filler is generated, and associating it with an ID (the *fid* is the filler fragment). Note that the filler can in turn have holes in it, which will be filled by other fillers.

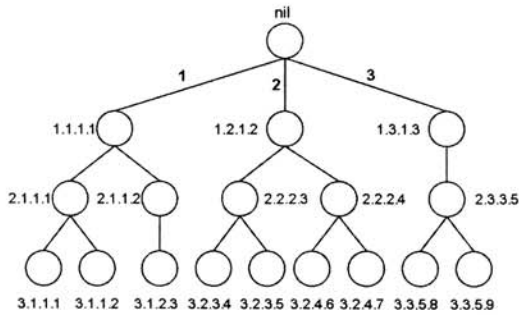
Also, the original XML document reconstruct by replacing holes with the corresponding fillers at the destination in the source. Attribute *tsid* (i.e., tag structure *id*) represents the ID of the fragment's root element in XML document DTD.

```
<stream: structure>
<tag name= "book" id= "1" Filler= "true">
    <tag name= "title" id= "2"/>
    <tag name= "authors" id= "3" Filler= "true">
        <tag name= "author" id= "4"/>
  </tag>
  <tag name= "year" id= "5"/>
  <tag name= "section" id= "6" Filler= "true">
      <tag name= "head" id= "7"/>
      <tag name= "subsection" id= "8" Filler= "true">
          <tag name= "head" id= "9"/>
          <tag name= "title" id= "10"/>
      </tag>
      <tag name= "figure" id= "11" Filler= "true">
          <tag name= "title" id= "12"/>
      </tag>
  </tag>
</tag>
</stream: structure>
```

(Fig. 3) Tag structure of Hole-Filler Model

3.2 A Novel Labeling Scheme

A robust labeling scheme supports the speedy inference of structure information of XML fragment such as parent-child relationship "/" and ancestor descendant relationship "//" between fragments. The proposed labeling scheme identifies structural relationships faster even if XML tree is deep; moreover, it has smaller label size than the prefix labeling scheme [15].



(Fig. 4) Labeled XML document

(1) Labeling construction

The labels of all nodes are constructed by the four significant components (*C1*, *C2*, *C3* and *C4*), which are unique.

(a) *C1* : It represents the level of node in XML document.

(b) *C2* : It represents the seed number of node which branches from root node. The seed number can apply from level *2* of leaf node that is increase *1*, respectively. The seed number of first leaf node is always *1*.

(c) *C3* : The component that is inherited from parent's label, which is *C4* of parent node.

(d) *C4* : It represents the relative location among sibling nodes.

A unique label is created by four components, which are concatenated by a "*delimiter* (.)". (Fig. 4) is the labeled XML tree by applying the definitions.

Definition 1. (Label for root node *r*) The root node does not have a level, a parent node and a sibling node, the value of root is *nil*.

Definition 2. (Label for internal node *x*) *C1* is represented by the level of corresponding node. *C2* is created by seed number which branches from root node. *C3* is created by inheriting *C4* (the parent's component). Finally, *C4* represents the order of sibling nodes. (x is a current node)

$$L(x) = C1_x \,.\, C2_x \,.\, C3_x \,.\, C4_x$$

(2) Speeding up structural relationships

The proposed labeling scheme provides an effective query processing by removing many redundant operations and minimizing the number of fragments being processed.

Seed labeling can quickly determine the ancestor-descendant relationship which only compares *C2*.

(Parent-childrelationship) If node x is a parent node of node y, the two labels satisfy the following.

(a) *C1* of parent node x is equal to *C1* + *1* of child node y.

(b) *C4* of parent node x is equal to *C3* of child node y.

(Ancestor-descendant relationship) If node x is an ancestor node of node y, the two labels satisfy the following.

(a) *C1* + *1* of descendant node y ≥ *C1* of ancestor node x

(b) *C2* of ancestor node x is equal to *C2* of descendant node y

(Sibling relationship) If node *x* is a preceding sibling node of node *y*, the two labels satisfy the following.

(a) *C1* of right node x is equal to *C1* of left node y

(b) *C3* of right node x is equal to *C3* of left node y

(c) *C4* of right node x is equal to *C4* +*1* of left node y

3.3 Fragment Representation

In Hole-Filler model [4], the fillers associate with holes by matching *fids* with *hids*. Given two XML fragments, it can not directly identify the ancestor-descendant (A-D) relationships between fragments. Due to these limitations, the required memory for query processing over streamed XML fragments would be increased. In order to solve this problem, the location information of fragments would be known. For this purpose, we extend the *fid* with the proposed labeling scheme, where the location of an element is represented as labeling (*C1*. *C2*. *C3*. *C4*). *C1* is the nesting depth of the fragment's root element in the original XML document.

(Fig. 5) represents four fragments of XML document in (Fig. 2) after coding *fid* and *hid* with the proposed labeling scheme. Here, root's filler (i.e., the root of fragment document) is *nil*. Other filler IDs can be generated by pre-order traversing XML document tree on the server side. The fillers is connected with holes by matching filler IDs with hole IDs. Fragment 4's *fid* corresponds to Fragment 3's *hid*. Fragment 4 is a subtree of Fragment 3 when XML document is reconstructed. For example, "*subsection*" and "*head*" are in the same filler and the level of this filler equals to the level of "*subsection*" in the original document in (Fig. 5).

Using the proposed labeling scheme, the fragment not only maintains the relationship information between correlated fragments, but also simply identifies descendant fragments according to seed number (*C2*). Thus, the

```
Fragment 1:
<stream: filler id="null" tsid="1">
<book name="corel">
  <title> XML </title>
  <stream: hole id="1.2.1.2" tsid="3"/>
  <year> 2007 </year>
  <stream: hole id="1.4.1.4" tsid="6"/>
  .....
</book>
</stream: filler>
```

```
Fragment 2:
<stream: filler id="1.2.1.2" tsid="3">
  <authors>
    <author> J. Lee </author>
  </authors>
</stream: filler>
```

```
Fragment 3:
<stream: filler id="1.4.1.4" tsid="6">
  <section>
    <head> XML DTD </head>
    <stream: hole id="2.4.4.7" tsid="8"/>
    .....
  </section>
</stream: filler>
```

```
Fragment 4:
<stream: filler id="2.4.4.7" tsid="8">
  <subsection>
    <head> ... </head>
    <title> ... </title>
    .....
  </subsection>
</stream: filler>
```

(Fig. 5) Example of XML fragments

proposed labeling scheme provides efficient processing of the descendant axis (//) and axis (/) of XPath expression. Given the label, parent child relationship between nodes obtains by matching their labels. For example, suppose that f_1 with label (f_1C1. f_1C2. f_1C3. f_1C4) and f_2 with label (f_2C1. f_2C2. f_2C3. f_2C4) are fragments. The f_2 is a descendant of f_1 iff $f_1C1 > f_2C1$ and $f_1C2 = f_2C2$. The f_2 is a child of f_1 iff $f_1C1 = f_2C1$ 1 and $f_1C4 = f_2C3$. Thus, Fragment 4 with label (*2.4.4.2*) is a child of Fragment 1 with label (*1.4.1.4*) in (Fig. 5).

〈Table 1〉 Query type

	XPath Expression
Q1	doc("book.xml")/book/section/subsection/title
Q2	doc("book.xml")/book/section[/figure/title]/section/title
Q3	doc("book.xml")/book/section[difficulty≧"default"]/title
Q4	doc("book.xml")/book/section//title

4. Performance Evaluation

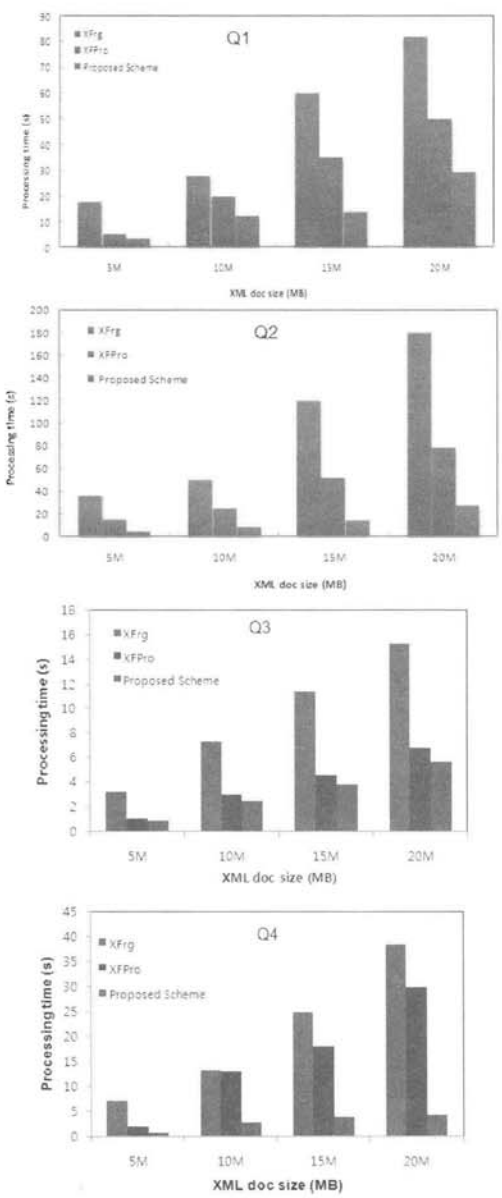
In our experimentation, it is assumed that XML document has been fragmented already. We focus on the query processing time and memory usage on streamed XML fragments on the client side. We have implemented XFrag [2], XFPro [6] and proposed scheme in Java on Windows XP Professional. We present the results of performance evaluation over different queries and document sizes. All experiments are conducted on a PC with 2.6GHz CPU and 2GB memory. The experiments are conducted on datasets generated by the Xmlgen of XMark benchmark [10]. We have used the following two queries on the generated "*book*" XML document and compared the results with XFrag [2], XFPro [6]. Table 1 shows the XPath expression used in the experiments. The memory usage was measured using the EclipseProfiler plugin [14] for the Eclipse IDE. The

book XML document was fragmented into *fillers* and *holes*, and the resulting filler fragments were processed sequentially.

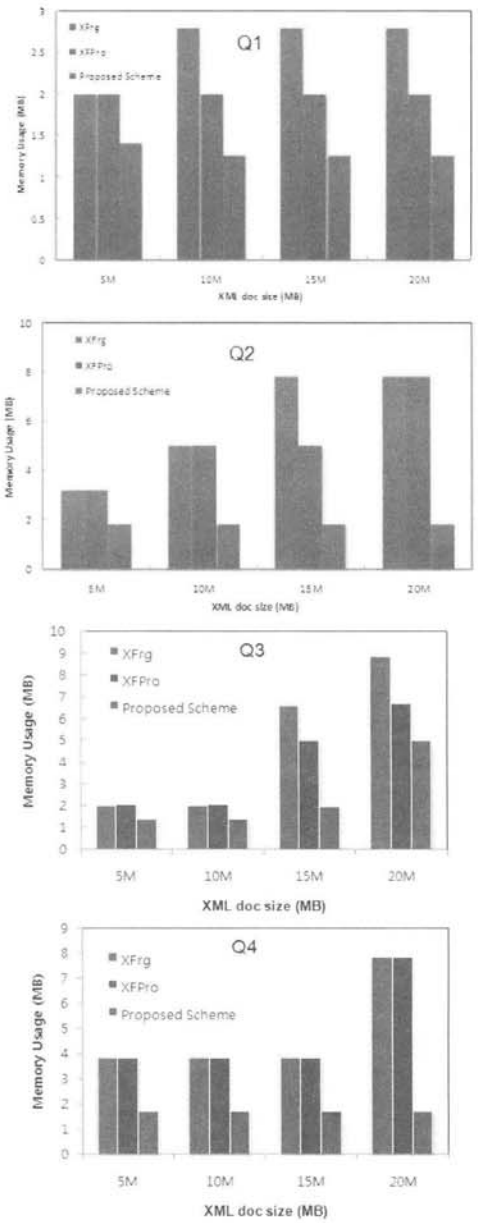
The "*section*", "*subsection*" and "*title*" are *filler* fragments according to the fragment information in tag structure, *Q1* processed "*subsection*" and "*title*" over fragment using seed label.

In XFrag, each fragment needs to be passed on through the pipeline and evaluated step by step. (Fig. 6) shows the query processing time over four queries. During query processing, the time was measured on the client side. In the experiments, the proposed scheme outperforms the other approaches. It is observed that XFPro outperforms XFrag mostly on query processing time. XFPro reduced CPU time which avoids subsumption operations for query processing time. However, it was not better than the proposed scheme because XFPro can not remove intermediate fillers when queries including "*" or "//". When the depth increases, the processing time of both XFrag and XFPro gets also poor due to intermediate fillers.

In (Fig. 6), XFrag is shown to be costly as the document size increased. In XFPro, the query processing pipeline is improved over XFrag because it just considers subroot nodes. However, many intermediate fillers occur for queries including "*" or "//". The proposed scheme shows the best performance for two queries because it removed redundant processing operations. Moreover, the proposed labeling scheme minimized the number of fragments being processed using *C2*. Comparing the proposed scheme with XFPro in terms of processing time,



(Fig. 6) Query processing time



(Fig. 7) Memory usage

the improvement ratio is *30%* (*Q1*) *40%* (*Q2*), *49%* (*Q3*) and *23%* (*Q4*) XFPro at 20MB XML document, respectively.

(Fig. 7) shows the memory usage for different document size. In the proposed scheme, memory usage is less sensitive to size since many intermediate fillers are removed. In XFrag and XFPro, memory usage gets worse with the increased size. However, XFPro performs better than XFrag because XFPro just considers subroot nodes while the XFrag deals with all operators in pipeline. Comparing the proposed scheme with XFPro in terms of memory usage, the improvement ratio is *30%* (Q1),*10%* (Q2), *15%* (*Q3*) and *23%* (*Q4*) at 20MB XML document, respectively.

5. Conclusion

This paper has proposed the novel labeling scheme for query processing over XML fragment stream using Hole-Filler model. The proposed labeling scheme can determine structural relationships between fragments very quickly. Thus, we can efficiently process structural relationships such as "//", "/", and branching nodes.

The conducted experimental results have shown that the proposed labeling scheme can reduce the query processing cost and memory usage cost for queries on XML fragments. We are currently working with energy efficient query processing and its evaluation.

References

[1] C. Barton, P. Charles, D. Goyal, M. raghavachari, M. Fontoura, and V. Josifovski, "Streaming XPath Processing with Forward and Backward Axes," Proc. of International Conference on Data Engineering, pp.455-466, 2003.

[2] S. Bose and L. Fegaras, "XFrag: A Query Processing Framework for Fragmented XML Data," Proc. of International Workshop on the Web Databases, pp.97-102, 2005.

[3] D. Carney et al., "Monitoring Streams-A New Class of Data Management Applications," Proc. of International Conference on Very Large Data Bases, pp.215-226, 2002.

[4] L. Fegaras, D. Levine, S. Bose, and V. Chaluvadi, "Query Processing of Streamed XML Data," Proc. of International Conference on Information and Knowledge Management, pp. 126-133, 2002.

[5] A. Gupta and D. Suciu, "Stream Processing of XPath Queries with Predicates," Proc. of ACM SIGMOD International Conference on Management of Data, pp.419-430, 2003.

[6] X. Hui, G. Wang, H. Huo, C. Xiao, and r. Zhou, "Region-Based Coding for Queries over Streamed XML Fragments," Proc. of International Conference in Web Information Systems Engineering, pp.487-498, 2006.

[7] B. Ludascher, Y. Papakonstantinou, and P. Velikhov, "Navigation-Driven Evaluation of Virtual Mediated Views," Proc. of International Conference of Extending Data Base Technology, pp.150-165, 2000.

[8] D. Olteanu, T. Furche, and F. Bry, "An Efficient Single-Pass Query Evaluator for XML Data Streams," Proc. of ACM Symposium on Applied Computing, pp.627-631, 2004.

[9] F. Peng and S. Chawathe, "XPath Queries on Streaming Data," Proc. of ACM SIGMOD International Conference on Management of Data, pp.431-442, 2003.

[10] A. Schmidt, F. Vaas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse, "XMark: A Benchmark for XML Data Management," Proc. of International Conference on Very Large Data Bases, pp. 215-226, 2002.

[11] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml.

[12] W3C Working Draft: XML Path Language (XPath), version 2.0 (2001) Technical Report WD-xpath20-20011220, W3C, 2001, http://www.w3.org/TR/WD-xpath2020011220.

[13] W3C Working Draft: XQuery 1.0: An XML Query Language. (2001) Technical Report WD-xquery-20010607, World Wide Web Consortium.

[14] http://eclipsecolorer.sourceforge.net/indexprofiler.html.

[15] Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System," Proc. of ACM SIGMOD International Conference on Management of Data, pp.204-215, 2002.

[16] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," Proc. of ACM SIGMOD International Conference on Management of Data, pp. 425-436, 2001.

[17] Jun-Ki Min, Myung-Jae Park, and Chin-Wan Chung, "XTREAM: An Efficient Multi-query Evaluation on Streaming XML Data," Vol.177, No.17, pp.3519-3538, 2007.

[18] Guoren Wang, Huan Huo, Donghong Han, and Xiaoyun Hui, "Query Processing and Optimization Techniques over Streamed Fragmented XML," World Wide Web, Vol.11, No.3, pp.339-359, 2008.

고 혜 경

e-mail : hkko@kaist.ac.kr

1999년 충북대학교 컴퓨터과학과(학사)

2002년 충북대학교 전자계산학과(이학석사)

2008년 고려대학교 컴퓨터학과(이학박사)

2008년~2009년 고려대학교 정보통신기술 연구소 연구교수

2009년~현 재 한국과학기술원 전산학과 연구교수

관심분야 : XML 데이터베이스, XML Security, 시맨틱 웹, Steam 데이터 관리, 온톨로지 관리, Service-Oriented Computing