Efficient Flash Retrieval Using Structural Join Index Hierarchy in an Object XML Framework

Hon Chung MAK Edmund[a], Chi-Wai FUNG[b], Qing LI[c]
[a]The Hong Kong Institute of Education, NT, Hong Kong, China;
[b]HKIVE (Tuen Mun) VTC, NT, Hong Kong, China;
[c]City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong, China

ABSTRACT

Flash movie is gaining widespread usage nowadays. There are dozen of Flash movies now available throughout the Internet. In order to allow Flash movies to be queried, we create an OXF (Object XML for Flash) model by converting Flash movies into XML format. We apply Structural Join Index Hierarchy (SJIH) indexing technique to enhance retrieval of OXF objects. A detailed cost model for query execution through SJIH is developed. Our experimental studies on using SJIH based on Flash movies has demonstrated performance advantageous when compared with other indexing techniques.

Keywords: XMKL, Object XML, SJIH, Rich Media Retrieval, Flash Movie Access

1. INTRODUCTION

The Web is experiencing a momentum growth of rich media contents, as exemplified by Flash which is a new format of vector-based interactive movie developed by Macromedia Inc. Flash movies are delivered over the Internet in the form of Macromedia Flash (SWF) file. As a matter of fact, Flash movies can also be converted into XML by, e.g., a utility such as JavaSWF2 [Mai02] for data transfer. A flash file can be treated as an encoded XML file (a flash file is binary while a XML file is ASCII text file).

XML, by itself, is flat file structure and is not suitable for query execution. By converting XML into Objects, resulting in what we call Object-oriented XML [ML02], we can enhance the query time by building index on it, making it more suitable for query than flat file.

In this paper, we present an approach for converting Flash into XML objects - an *Object XML for Flash (OXF)* framework. On top of the framework, we apply the *Structural Join Index Hierarchy (SJIH)* technique to enhance the retrieval of XML objects. Detailed studies are also be conducted to demonstrate the effectiveness of the SJIH indexing approach.

The structure of the rest of the paper is as follows: Section 2 briefly reviews the background of our research. In section 3, we describe the processes of objectifying XML. Using Flash movies as the examples, Section 4 discusses how Object-XML is used with respect to pre-processing and query handling, and Section 5 describes the performance study through the SJIH cost model. Finally, we conclude the paper and offer further research directions in section 6.

2. BACKGROUND OF RESEARCH

In this section, we review the basic aspects of our research in terms of Flash, JavaSWF2 and SJIH and Object-Oriented XML.

2.1 Flash

Flash is a vector-based interactive movie created by using Macromedia Flash tools. It is gaining wide-spread popularity in this Internet Information age. It has three major aspects [YLW+02]:

- It contains heterogeneous components, e.g. texts, graphics and images. All these components are encoded separately so that they can be easily extracted from Flash data files.

Electronic Imaging and Multimedia Technology IV, edited by Chung-Sheng Li, Minerva M. Yeung, Proc. of SPIE Vol. 5637 (SPIE, Bellingham, WA, 2005)
0277-786X/05/$15 · doi: 10.1117/12.569996

- It features dynamic effect. A flash movie is constituted by a sequence of frames that are played in an order subject to user interactions.
- Its interactivity allows users to interact with the movie. Unlike a passive media such as a video stream, user can interact with Flash movie and hence can interfere the presentation of the movie.

2.2 JavaSWF2

JavaSWF2 is a set of Java packages that enable the parsing, manipulation and generation of the Macromedia Flash(TM) file format known as SWF [Mai02]. JavaSWF2 can convert a Flash movie into XML and vice versa. Figure. 1 shows an example Flash movie and its corresponding XML file (as given in Figure. 2). It describes a car that is moving from left to right horizontally.

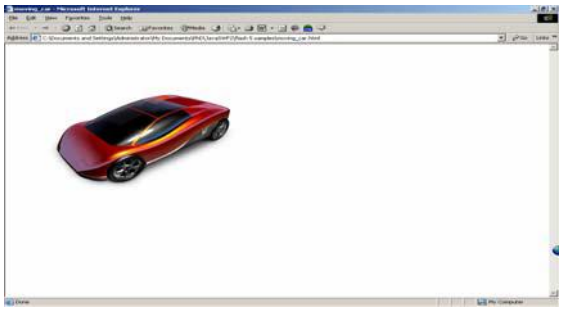
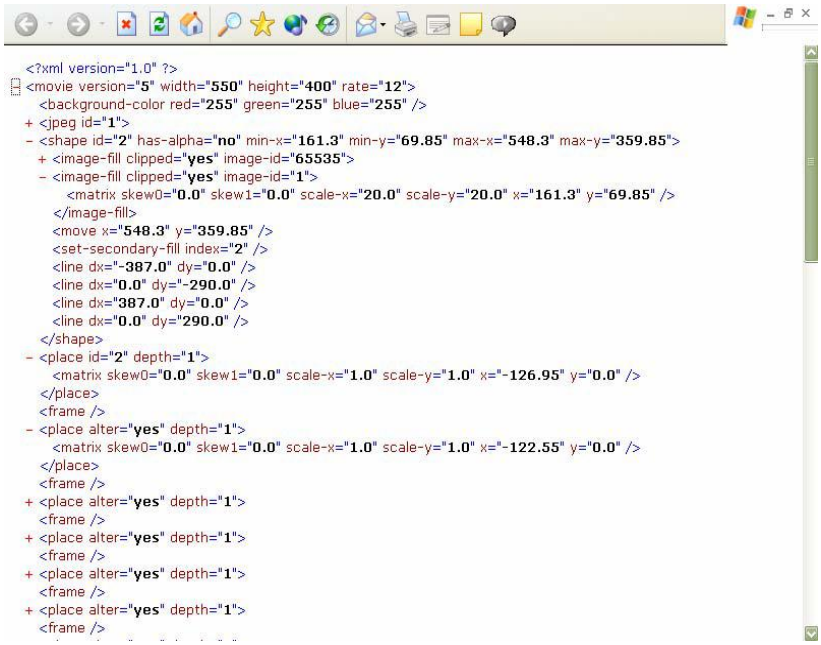


Figure 1: A moving car Flash file.

2.2 Structural Join Index Hierarchy (SJIH)

There has been some work done on indexing techniques for efficient query processing along a class composition hierarchy (CCH). These techniques include: Multi-index (MI) [MS86], Join index (JI) [Val87], Nested index (NI) [BK89], Path index (PI) [BK89], Access Support Relation (ASR) [KM90] and Join index hierarchy (JIH) [XH94]. By comparing the previous work on indexing methods, we note the following points: (a) In terms of the number of index lookups required for query processing, NI, PI, ASR and JIH are better than MI and JI. (b) MI, NI and PI only support reverse traversal, but JI, ASR and JIH support traversals in multiple directions. (c) All of the above indexing methods index on classes along a single path, hence for complex object retrieval that requires results from classes on multiple paths, extra processing is required. In contrast, our SJIH framework supports multiple path queries, facilitates traversal in both forward and reverse direction, and at the same time is extensible for accommodating new user requirements.



```
<?xml version="1.0" ?>
<movie version="5" width="550" height="400" rate="12">
  <background-color red="255" green="255" blue="255" />
+ <jpeg id="1">
- <shape id="2" has-alpha="no" min-x="161.3" min-y="69.85" max-x="548.3" max-y="359.85">
  + <image-fill clipped="yes" image-id="65535">
  - <image-fill clipped="yes" image-id="1">
      <matrix skew0="0.0" skew1="0.0" scale-x="20.0" scale-y="20.0" x="161.3" y="69.85" />
    </image-fill>
    <move x="548.3" y="359.85" />
    <set-secondary-fill index="2" />
    <line dx="-387.0" dy="0.0" />
    <line dx="0.0" dy="-290.0" />
    <line dx="387.0" dy="0.0" />
    <line dx="0.0" dy="290.0" />
  </shape>
- <place id="2" depth="1">
    <matrix skew0="0.0" skew1="0.0" scale-x="1.0" scale-y="1.0" x="-126.95" y="0.0" />
  </place>
  <frame />
- <place alter="yes" depth="1">
    <matrix skew0="0.0" skew1="0.0" scale-x="1.0" scale-y="1.0" x="-122.55" y="0.0" />
  </place>
  <frame />
+ <place alter="yes" depth="1">
  <frame />
+ <place alter="yes" depth="1">
  <frame />
+ <place alter="yes" depth="1">
  <frame />
+ <place alter="yes" depth="1">
  <frame />
```

Figure 2: A moving car XML file.

In our SJIH framework, we use the join index approach to store OIDs in the form of tuples for efficient complex object retrieval. In the SJIH framework, we present the following index categories: (1) base join index (BJI) ([XH94]), (2)

derived join index (DJI) ([XH94]), (3) hyper join index (HJI) (our contribution), and (4) structural join index (SJI) (our contribution). Furthermore, BJI and DJI are defined over a single path, whereas HJI and SJI are defined over multiple paths.

The most primitive index under SJIH is the BJI: a BJI is used to support navigation between two adjacent classes. A BJI is a binary relation with the two attributes being the OIDs of two neighbouring classes. Furthermore, BJIs are also useful in building up more complex higher level indices.

A DJI is a ternary relation with the first two attributes being the OIDs of two classes along the same path, and the third attribute (*dup*) is a duplicating factor which records the number of duplicates. As in [XH94], a value greater than 1 means that there is more than one connection relating the OIDs of two classes in the OODB. These duplicate factors facilitate the management of the impact of object deletions on the DJI. DJIs are "derived" from the BJIs. While a BJI relates two adjacent classes in the OODB schema, a DJI relates classes that are one or more classes apart and on the same path. In a more complex OODB schema, a DJI can also be derived from other DJIs and/or BJIs.

A HJI is also a ternary relation, where the first two attributes are the OIDs of two classes (over two paths). The third attribute is still the duplicating factor. HJIs also relate objects of two classes in an OODB schema and consists of two OIDs. But HJI can involve classes from two different paths. Note that a DJI is a special case of a HJI when the two classes are from the same path.

For a SJI which indexes on n classes and hence containing a total of n OIDs, the SJI is an $n+1$-ary relation, where the first n attributes are the OIDs of the classes. The last attribute is still the duplicating factor. An SJI is a sophisticated structure as it may contain more than two OIDs from classes in an OODB schema. In fact, an SJI may even cover the whole schema structure (complete SJI), though more often it covers part of the schema structure (partial SJI). From the design point of view, an SJI is a high level abstraction of the original OODB schema that contains the OID information about the classes that are indexed by the SJI.

In a query processing environment with multiple queries, one single indexing method is often not enough to expedite all queries. Furthermore, the user requirements and query patterns may change, requiring accesses to different classes during query processing. Our SJIH is a framework designed to this problem. In particular, a SJIH can consist of a set of partial SJIs (or even a complete SJI) coupled with simpler BJIs, DJIs and HJIs. An ideal SJIH should expedite most of the queries in the query processing environment. When the query pattern changes, we can use the existing indices in the SJIH to form new indices. The following classifications of SJIHs are introduced for a given OODB schema:

- **Complete-SJIH (*CSJIH*)**: It is a single large SJI that covers every class in the schema. Complete-SJIH is the most powerful as all the OID information and hence all the relationships between objects in all the classes are captured. The more the number of classes a SJIH has, the more *complex* it is. The major problem with Complete-SJIH is its large storage and update overhead, especially for a high degree of sharing between composite objects and their component objects.
- **Partial-SJIH (*PSJIH*)**: A Partial-SJIH may contain a number of partial SJIs and some other simpler BJIs, DJIs and HJIs. Since not all the OID information is stored but spread over a number of indices, we may need to combine these indices to obtain the results for query execution. Partial-SJIH has the advantage of lower update cost, but it may imply higher query processing cost. For cases with a high degree of sharing, the storage overhead will be much lower than that of the Complete-SJIH.
- **Base-SJIH (*BSJIH*)**: It is the collection of *all* the BJIs between every two neighbouring classes in the OODB schema. Query processing requires combining the relevant BJIs to obtain the result. The main advantage of the Base-SJIH is its low storage overhead even in the case of a high degree of sharing. Another advantage is that it is efficient to update a BJI. The main problem with the Base-SJIH is that the performance of query processing may not be as good as that of the Complete-SJIH or a Partial-SJIH, due to the large number of BJIs involved.

2.4 Object-Oriented XML

Due to the similarity of XML structure to that of objects in an object-oriented database (OODB), it is more natural to represent it in OODBs [Mer01]. By incorporating attributes and methods into XML objects, the functionality of XML can be further enhanced. To this end, an object schema such as the one shown in Figure 3 is needed, based on which XML elements are annotated and converted into XML objects.

2.4.1XML to OO Mapping

As an example, we consider the following XML as given in Figure 4. By annotation, we can translate it into an OO object as shown in Figure 5.

Each DDL file is processed and analyzed, and enhanced features are added (manually) if necessary and a XML file is generated. Methods, such as those for executing system's function calls, can be incorporated into an XML object. For example, if an object contains a video clip, the object's function invokes windows media player and *display* its content to the media player. If an object contains MP3 audio clip, it can invoke WinAmp and "play" the music automatically. In our work, the methods are incorporated as part of the XML objects to enhance the query on Flash media data, with the possibility of increasing the performance of the system.

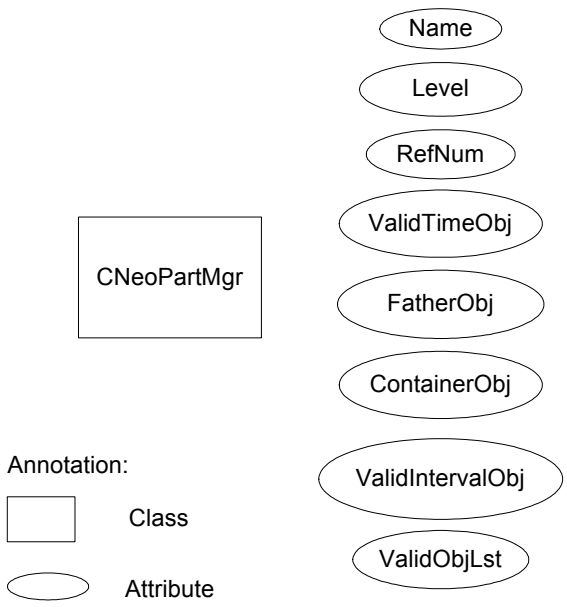


Figure 3: An object schema

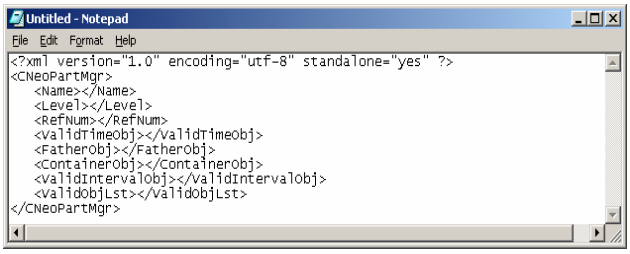


Figure 4: Example of XML.

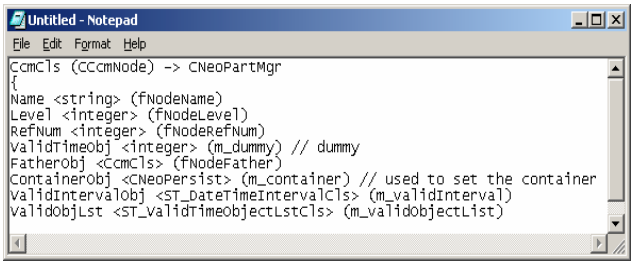


Figure 5: Example of OO Object.

3. THE OXF FRAMEWORK

To show the relevance and usefulness of Object-Oriented XML, we present the following *Object-XML for Flash* (OXF) framework to illustrate our approach. The example application is an extension of the research project of FLAME [YLWZ02], a web-based Flash retrieval system. The processes involved are elaborated in the subsequent subsections.

3.1 Storing Process

(1) Flash file is converted into XML format by using JavaSWF2 (which is a Flash to XML converter. A XML file is then generated which has two categories of tags: definition tags and control tags. Definition tags are used to define various components in a movie, and control tags, which are used to manipulate these components to create the dynamic and interactive effect of the movie,(e.g., DefineShape and DefineText and typical definition tags), while PlaceObject and ShowFrame are typical control tags.

(2) XML elements are extracted out and converted into a number of OO objects based on direct mapping of XML tag with Objects. These objects are then input into the System.

(3) The OO objects, once created, can be indexed by SJIH among the component objects and stored in the OODB.

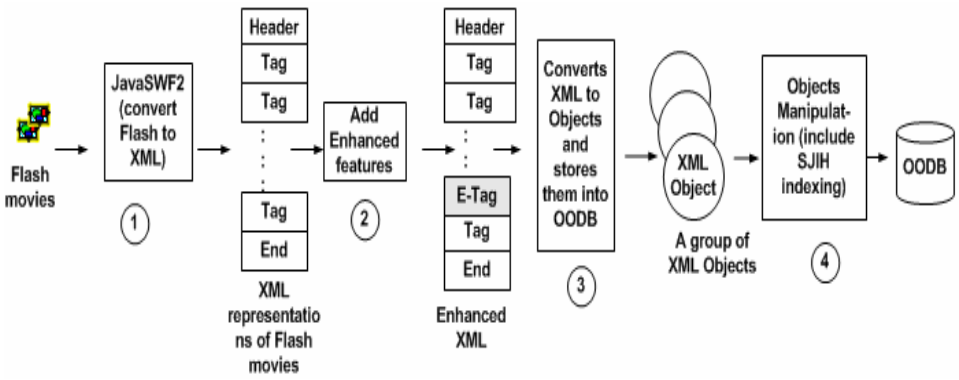


Figure 6: OXF Framework.

3.2 Querying Process

Users enter object-XML queries through the Query Interface to the OODB (cf. Figure 7). Objects are retrieved by the SJIH index. Retrieved results are then passed to an ***OO to XML Converter***. This process converts the OO objects into XML format and passes the XML results to the XML parser. The parser interprets the results and displays them in a readable format to the user.

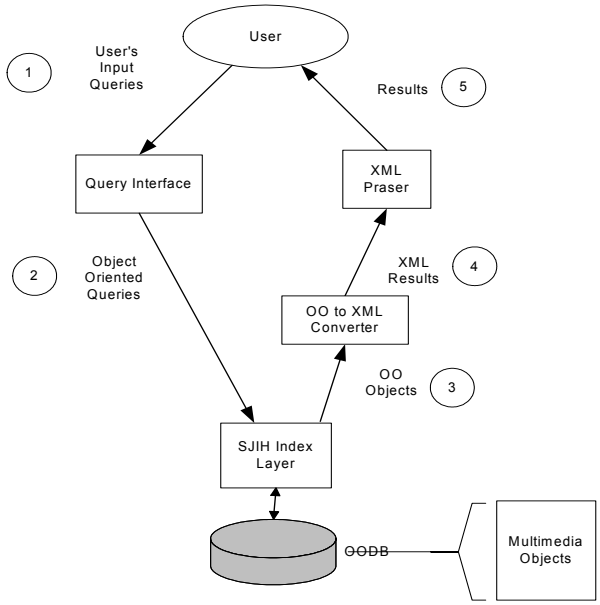


Figure 7: Querying XML Object

4. SJIH FRAMEWORK[FL01]

4.1 Storage cost for evaluating SJIH

We store OIDs of a SJI in the form of tuples in the leaf level of a clustered B^+ tree index. (We can also build non-clustered indices on the SJI tuples if required.) The index tuple consists of OIDs of classes on which the SJI is defined. The tuples are stored as a relation with no redundancy. If there are duplicate SJI tuples, a "duplicating factor" is used to record the total number of duplicates. This duplicating factor can facilitate the management of the index tuples. Let us consider a generic SJI with n tuples which occupies m pages of storage as shown in Figure 8. Assume this SJI has a clustered index on OIDs of class $C1$; furthermore, assume it has non-clustered indices on OIDs of the classes $C2$, ... , CX. The storage cost of this SJI is then:

Storage Cost = Storage Cost for Clustered Index + Storage Cost for Non-Clustered Indices.

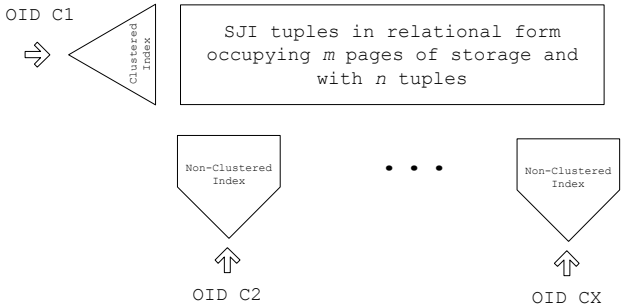


Figure 8: SJIH with clustered and non-clustered index

As in [XH94], we concentrate on the storage cost of the *leaf-level nodes* of the clustered index that stores the SJI tuples. We neglect the storage cost for the *non leaf-level nodes* of the clustered index as it is small when compared with the leaf-level. Furthermore, as not every class in a SJI has a non-clustered index on it, for uniform comparison between different SJIs, the storage costs for non-clustered indices are also neglected. The storage cost of the SJI is given by: $m = \left\lceil \frac{n \times STJI}{PS \times POF} \right\rceil$ where n = the number of SJI tuples, STJI = length of SJI tuples, PS = page size of the system, POF = page occupancy factor of the B^+ tree index.

4.2 Index retrieval cost for evaluation SJIH

For calculating the index retrieval cost, there are two cases to be considered separately: Clustered index and Non-clustered index.

4.2.1 Clustered index

To estimate the number of disk accesses to a collection of leaf/non leaf nodes, we use the Yao function [Yao77]. This function is used because data records are grouped into blocks in secondary storage and to estimate the number of blokcs accessed for a given query is essential. Specifically, given n nodes uniformly distributed into m pages ($1 < m \le n$), if k nodes ($k \le n$) are randomly selected from these n nodes, the expected number of disk accesses is given by:

$Yao(k,m,n) = m \times \left[1 - \prod_{i=1}^{k} \frac{nd - i + 1}{n - i + 1} \right]$, where $d = 1 - \frac{1}{m}$. (The expected number of disk accesses is not equal to k because some pages may contain two or more nodes.) But the Yao function only applies when $m \le n$, i.e., when a node is smaller than or equal to the page size. For larger nodes, we estimate the number of disk accesses by a simple proportion: $m \times k / n$. In building the cost model, we thus use an auxiliary function Y as defined below:

$$Y(k,m,n) = \begin{cases} Yao(k,m,n) \\ \quad Case1 : for\, node\, size \le page\, size \\ m \times k / n \\ \quad Case2 : for\, node\, size > page\, size \end{cases}$$

For a SJI occupying m pages of storage and with n tuples, if we want to retrieve k tuples out of these n tuples, the index retrieval cost is then:

Index retrieval cost = Disk access for leaf-level nodes + Disk access for non leaf-level nodes.

Assuming an OODB system with PS=8192 bytes, average POF of 70% and BTF (average fan-out of the B^+ Tree) of 409, the two disk access costs are calculated as follows: (a) Disk access for leaf-level nodes is $Y(k,m,n)$. (Note that the leaf-level nodes store the SJI tuples.) (b) Disk access for non leaf-level nodes is $Y(Y(k,m,n), \left\lceil \frac{m}{BTF} \right\rceil, m)$.

4.2.2 Non-clustered index

For non-clustered index, the index retrieval cost is:

Index retrieval cost = Disk access for leaf-level nodes of the clustered index + Disk access for non leaf-level nodes of the non-clustered index.

Note that non-clustered index does not have its own SJI tuples; rather, it shares the leaf-level nodes with the clustered index. The two disk access costs are calculated as follows: (a) Disk access for leaf-level nodes of the clustered index is $Y(k,m,n)$. (b) Disk access for non leaf-level nodes of the non-clustered index is: $Y(k,\left\lceil \frac{n}{BTF} \right\rceil,n) + Y(Y(k,\left\lceil \frac{n}{BTF} \right\rceil,n),\left\lceil \frac{\left\lceil \frac{n}{BTF} \right\rceil}{BTF} \right\rceil,\left\lceil \frac{n}{BTF} \right\rceil)$.

5. PERFORMANCE EVALUATION

Performance is a key factor for the success of object oriented database (*OODB*) systems, especially when supporting semantically rich database applications such as engineering and multimedia database applications. For these applications complex object retrieval has a major impact on the cost of processing the queries [KL95,KL00,FL01]. One way to improve the performance of complex object retrieval is to use indices. To demonstrate the validity and effectiveness of our proposed OXF framework, we proceed to evaluating the Flash query processing performance using SJIH versus other schemes like Multi-index, Nested index, and Access Support.

5.1 Experimental setup

In the racing cars Flash movie example shown in Figure 9, two cars are moving from left to right with different speed. They arrive at the destination at different time frame. Car 2 arrives the destination before Car 1 and moves faster than Car 1. To reduce the complexity, this movie has been skimmed to contain only two frames. By converting the Flash movie into OXF, we have the classes in Figure 10. We use the racing cars example schema for this evaluation to evaluate the utility of the following three SJIHs:

- Complete-SJIH (CSJIH) -- the complete SJIH, a single SJI that covers all seven classes of the schema.
- Base-SJIH (BSJIH) -- the SJIH with *all* the BJIs between every pair of adjacent classes. There will be a total of six BJIs in the BSJIH.
- Partial-SJIH (PSJIH) -- a partial SJIH (as shown in Figure 11) that covers five objects in the schema.

For Multi-index, we build a total of six MIs to associate every two neighbouring classes. For Nested index, we build four NIs, one NI for each of the following class pairs: *movie~Car1.frame1, movie~Car1.frame2, movie~Car2.frame1,* and *movie~Car2.frame2.* For ASR, we build four ASRs, one ASR for each of the following class groups: *movie~Car1~Car1.frame1, movie~Car2~Car2.frame2, Car1~Car1.frame2*, and *Car2~Car2.frame1*. For SJIH, we use the three SJIHs.

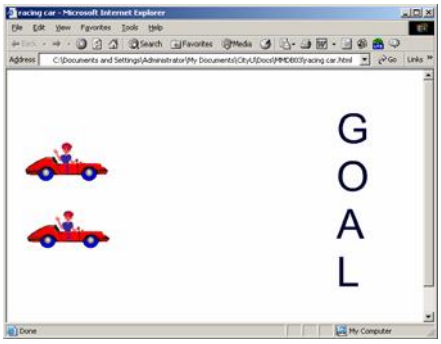


Figure 9: Racing car example.

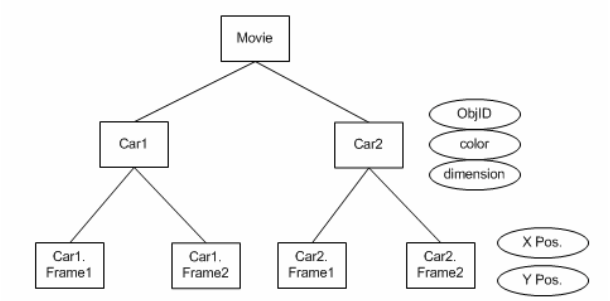


Figure 10: Example of racing cars classes instance

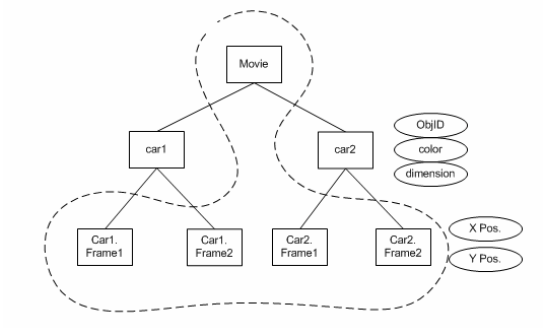


Figure 11. Example of partial -SJIH

5.2 Experimental result and explanation

From the cost model, the *degrees of sharing* between composite objects and their component objects (i.e., the forward fan-out and the reverse fan-out values) are the most influential factors on the index retrieval cost. In this experiment, we set both fan-outs to the *scale factor*. In this evaluation, the scale factor range is: 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8 and 16.

From Figure 12, at low to medium scale factor (0.1 - 1.0), CSJIH is the lowest in terms of storage cost required, implying that CSJIH is the best candidate as the indexing method in this scale factor range. The storage costs are in the following order: MI > BSJIH> ASR > NI > PSJIH > CSJIH. At a high scale factor range (say >1.0), CSJIH will not be the best indexing method. The storage costs are in the following order: CSJIH > ASR > PSJIH > NI > BSJIH > MI.

At low scale factor, PSJIH and CSJIH are the best in terms of storage cost requirement. As the scale factor increases, the storage costs of all the indexing methods also increase. As the scale factor further increases, MI and BSJIH become the best while CSJIH's storage cost keeps increasing and becomes the highest storage cost.

Note that a low scale factor means the forward fan-out values are also low, which implies that there are very few object connections between the objects in classes *movie, Car1.frame1, Car1.frame2, Car2.frame1* and *Car2.frame2*. Due to the multiplying effect in the forward fan-out values along the paths, the more complex SJIHs have the lower storage cost at low scale factor. As MI and NI involve only one path in the schema, the multiplying effect in the forward fan-out values is not as prominent as that on the multi path SJIHs, so they will have higher storage cost at low scale factor. But when the scale factor is larger than 1, the multiplying effect in the number of object connections causes more complex SJIHs' storage costs to increase rapidly. One conclusion is that since BJI and DJI/HJI serve similar functionality as Multi-index and Nested index, we should employ SJIH with simpler BJIs, DJIs and HJIs in cases of high degree of sharing.

From figures 13 to 16, we show the plots of Scale factor vs. Index retrieval cost for the different indexing methods with selectivities at 0.001, 0.01, 0.1 and 1.0, respectively. We focus on Figure 13 first. From the cost model, the index retrieval cost (c.f. section 4.1.2) has a high correlation with the storage cost, hence we see a very similar pattern for the variations of the index retrieval cost and the storage cost as in Figure 12.

We compare the four result plots in Figures 13 to 16. When the selectivity is higher than a particular threshold, the index retrieval cost will be independent of the selectivity. As selectivity of 0.1 and 1.0 are above the threshold, the result plots in Figures 15 and 16 are identical, showing that the index retrieval cost is independent of the selectivity.

From Figures 13 to 16, we can notice that MI, NI and BSJIH indexing methods are unsuitable for scale factors less than 1, because there are other indices (PSJIH and CSJIH) which outperform MI/NI by a factor of 100. For large scale factors, the only SJIH that provides as good performance as MI/NI is BSJIH. Depending upon the scale factor, we can decide on the most effective SJIH judiciously.

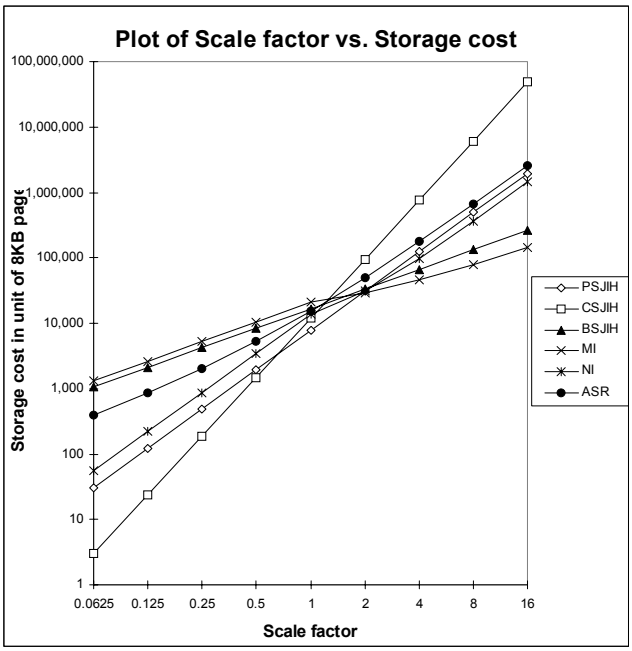


Figure 12. Plot of Scale Factor Vs Storage Cost.

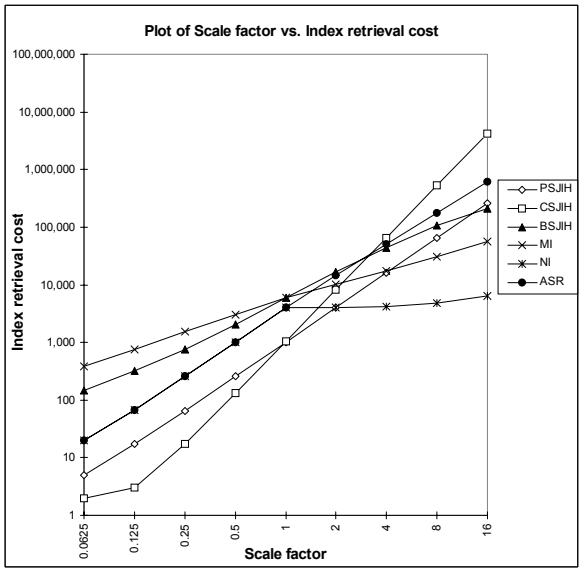


Figure 13. Plot of Scale factor vs. Index retrieval cost at selectivity 0.001.

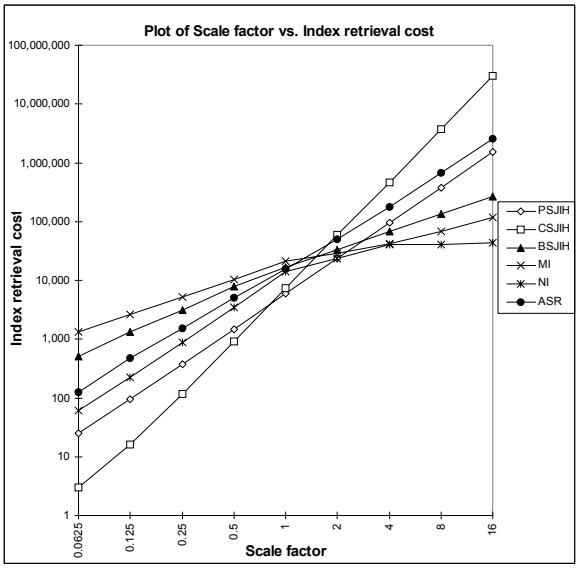


Figure 14. Plot of Scale factor vs. Index retrieval cost at selectivity 0.01.

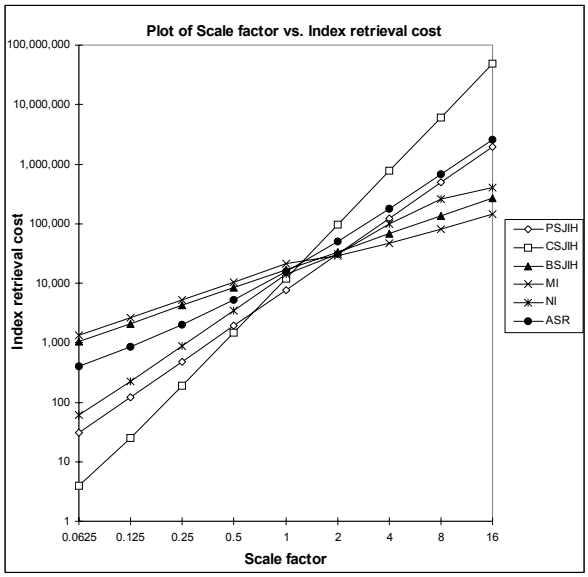


Figure 15. Plot of Scale factor vs. Index retrieval cost at selectivity 0.1.

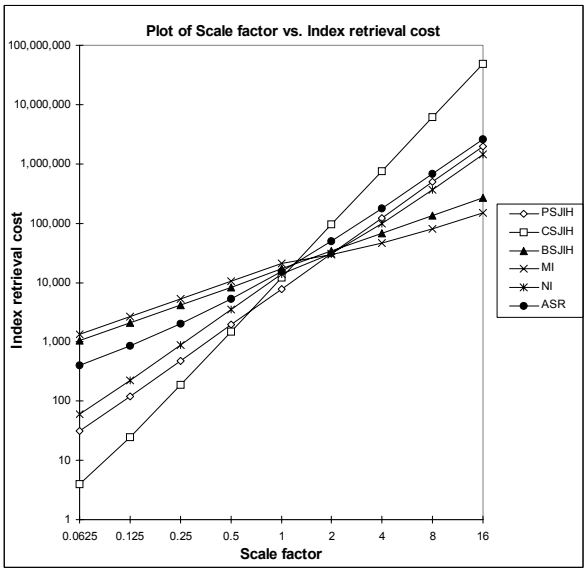


Figure 16. Plot of Scale factor vs. Index retrieval cost at selectivity 1.0.

5. CONCLUSION AND FUTURE WORK

In this paper, we have described the basic aspects of our OXF framework of storing and managing XML data based on an object-oriented database. Our approach has shown the effectiveness and advantages in supporting advanced rich media applications such as Flash movie retrieval. Since the tasks of parsing and manipulating Flash are closely tied with and dependent on XML, our work of extending and handling XML in an object-oriented way provides a feasible approach to enhance the performance so that it can facilitate rich media retrieval more efficiently.

Our approach demonstrates that XML can be naturally modeled as objects and can be queried. The SJIH indexing techniques further contribute to solving the complex object retrieval problem often encountered in processing large amount of objects. From the experimental results, it is show that the performance of using SJIH has been advantageous when compared with other indexing techniques.

In our future work, extensive studies will be performed on investigating the object XML query processing and optimization issues. Also, further development of the prototype system's capabilities and facilities, including spatial-temporal modeling and queries, indexing and view mechanisms, will be conducted and tested upon a full-scale Flash movie application domain. Finally, to make the proposed ideas more practical for large scale Flash movie, automation of the process is essential and should be further explored

REFERENCE

[**BK89**] E. Bertino and W. Kim, "Indexing Technique for Queries on Nested Objects", in *IEEE Trans. on Knowledge and Data Engineering,* 1(2): 196-214, 1989.

[Bou00] Ronald Bourret, “XML and Databases”, http://www.rpbourret.com/xml/XMLAndDatabase.htm, November, 2000

[Bos99] Bert Bos, http://www.w3.org/XML/1999/XML-in-10-points, July, 2001

[Car75] Cardenas, A.F, “Analysis and performance of inverted database structures”, *Comm. ACM 18*, 5 (May 1975), 253-263

[CL00] Shermann S.M. Chan, Qing Li, “Architecture and Mechanisms of a Web-based Video Data Management System”, *(CD-ROM) Proc. IEEE Int'l Conference on Multimedia and Expo (ICME'2000)*, New York, USA, July 2000

[Day01] Neil Day, “MPEG-7 Projects and Demos”, *ISO/IEC JTC1/SC29/WG11 N4034*, Singapore, March 2001

[Exp01] http://www.expway.tv/tutorial/xmlmpeg7/powerpoint.html

[FK99a]Daniela Florescu, Donal Kossmann. "Storing and Querying XML data using an RDBMS", *Bulletin of the Technical Committee on Data engineering, IEEE Computer Society,* September 1999, Vol22 No. 3, 1999
[KL95] K. Karlapalem and Q. Li, "Partitioning schemes for object oriented databases", Proc. of the Fifth Intl. Workshop on Research Issues in Data Engineering - Distributed Object Management (RIDE-DOM'95), Taipei, Taiwan, pages 42-49, 1995.
[KL00] K. Karlapalem and Q. Li, "A Framework for Class Partitioning in Object-Oriented Databases", Distributed and Parallel Databases Journal, 8, pages 317-350 (2000).
[FK99b] D. Florescu and D. Kossmann. "A performance evaluation of alternative mapping schemes for storing XML data in a relational database", *Technical Report, INRIA,* France, 1999.
[FKL00] Chi-Wai Fung, Kamalakar Karlapalem, Qing Li. "Complex Object Retrieval via Structural Join Index hierarchy Mechanisms: Evaluation and Selection Approaches", *Proc. 9th ACM Int'l Conference on Information and Knowledge Management (CIKM'00)*, Washington DC, Nov.6-11, 2000
[FL01] Chi-wai FUNG, Qing LI. "Efficient Multimedia Database Indexing Using Structural Join Index Hierarchy". *IEEE PCM'2001*, Beijing.
[ISO98] "MPEG-7 Requirements Document V.7", *ISO/IED JTC1/SC29/WG11/N2461 MPEG98,* October 1998, Atlantic City, USA.
[JM02] Hasan M. Jamil, Giovanni A. Modica. "An Object-Oriented Extension of XML for Autonomous Web Applications", *ACM CIKM'02*, November 409, 2002, McLean, Virginia, USA, ACM.
[**KM90**] A. Kemper and G. Moerkotte, "Access Support in Object Bases", in *ACM-SIGMOD Intl. Conf. on Management of Data*, Atlantic City, N.J., p364-374, 1990.
[LL01] Grace Wai-yue Leung, Qing Li, "Database as Virtual XML Documents: an Interoperable Approach" *Proc. of National Database Conference (NDBC'2001)*, Hebei, China, August 2001.
[LCWZ01] Qing Li, Shermann S.M. Chan, Yi Wu, and Y. Zhuang, "Web-based Video Database Management: Issues, Mechanisms, and Experimental Prototype" *Proc. of Int'l Conference on Distributed Multimedia Systems (DMS'2001)*, pp. 94-100, Taipei, Taiwan, Sept 2001.
[Mai02] D. Nick Main, http://anotherbigidea.com/javaswf
[Mar01]Jose M. Martmez, "Overview of the MPEG-7 Standard (version 5.0)", *ISO/IEC JTC1/SC29/WG11 N4031,* Singapore, March 2001.
[Mer01]David Mertz, "XML Matters # - Putting XML in context with hierarchical, relational, and object-oriented-models", www.106.ibm.com/developerworks/xml/library/x-atters8/index.html
[ML02] Hon Chung Mak, Qing Li: An Object XML-based Approach for Multimedia Application Processing and Development. JCIS 2002: 1041-1044
[**MS86**] D. Maier and J. Stein, "Indexing in an object-oriented DBMS", in *Proc. IEEE Intl. Workshop on Object-oriented Database System*, p171-182, Asilomar, Pacific Grove, CA, September 1986.
[Sal00] Philippe Salembier, "Status of MPEG-7: the Content Description Standard", *International broadcasting Conference,* Amsterdam, The Netherlands, September 8, 2000.
[SFC00]Cosima Schmauch, Torsten Fellhauer, Sissi Closs, "A Comparison of Database Types for Storing XML Documents", University of Applied Science, Karlsruhe, Business Information Systems Department, P.O. Box 2440, 76012 Karlsruhe, Germany.
[**Val87**] P. Valduriez, "Join indices", in *ACM Trans. on Database Systems*, 12(2):218-246, 1987. .
[**XH94**] Z. Xie and J. Han, "Join Index Hierarchies for Supporting Efficient Navigations in Object-Oriented Databases", in *Proc. Intl. Conf. on Very Large Data Bases,* p522-533, 1994.
[Vel00] Daan Velthausz, "Content Description MPEG-7 and MPEG-21", *European Middleware Workshop*, Telematica Institute, The Netherlands, 20 June, 2000.
[Vet01] Anthony Vetro, "MPEG-7 Applications Document V.10", *ISO/IEC JTC1/SC29/WG11/N3934,* January 2001/Pisa
[Yao77] S.B. Yao. "Approximating Block Accesses in Database Organizations", *Comm. of the ACM*, 20(4):260, April, 1977.
[YHC02] Shanzhen Yi, Bo Huang, Weng Tat Chan, "Spatio-Temporal Information Integration in XML", *Proc. 3rd International Conference on Web Information Systems Engineering* (WISE 2002 Workshop on Web and Wireless Geographical Information Systems), pp. 103-112, IEEE Computer Society, Singapore, Dec. 2002.
[YLW+02] Jun Yang, Qing Li, Liu Wenyin Yueting Zhuang. "Search for Flash Movies on the Web", *Proc. 3rd International Conference on Web Information Systems Engineering Workshops* (WISE 2002 Workshop on Mining for Enhanced Web Search), pp. 257-266, IEEE Computer Society, Singapore, Dec. 2002.