XML Data Integration and Distribution in a Web-based Object Video Server System

Shermann Sze-Man Chan[1], and Qing Li[2]

[1] Department of Computer Science,
[2] Department of Computer Engineering and Information Technology,
City University of Hong Kong,
83 Tat Chee Avenue, Kowloon, Hong Kong SAR, China
{shermann@cs., itqli@} cityu.edu.hk

Abstract. Data integration and distribution, albeit "old" topics, are necessary for developing a distributed video server system which can support multiple key functions such as video retrieval, video production and editing capabilities. In a distributed object video server (DOVS) system, objects from (homogeneous and heterogeneous) servers usually need to be integrated for efficient operations such as query processing and video editing. On the other hand, due to practical factors and concerns (such as resource cost and/or intellectual property concerns), raw/source video files often need to be well protected. XML is becoming the standard for multimedia data description (e.g. MPEG-7), and is very suitable for Web-based data presentation owing to its expressiveness and flexibility. In this paper, we describe our approach to process XML descriptions for data integration and distribution in a Web-based DOVS system.

1 Introduction

Web-enabled multimedia search and management systems become important in multimedia information management. Video is a rich and colorful media widely used in many of our daily life applications like education, entertainment, news spreading etc. Digital videos have diverse sources of origin such as tape recorder and Internet. Expressiveness of video documents decides their dominant position in the next-generation multimedia information systems. Unlike traditional types of data, digital video can provide more effective dissemination of information for its rich content.

Collectively, a digital video can have several information descriptors: (i) media data - actual video frame stream, including its encoding scheme and frame rate; (ii) metadata - information about the characteristics of video content, such as structural information and spatio-temporal features; (iii) semantic data - text annotation relevant to the content of video, obtaining by manual or automatic understanding. Metadata is created independently from how its contents are described and how its database structure is organized later. It is thus natural to define "video" and other meaningful constructs such as "scene", "frame" as objects corresponding to their respective inherent semantic and visual contents. Meaningful video scenes are identified and associated with their description data incrementally. Depending on user's viewpoint, same video/scene may be given different descriptions. On the other hand, XML is becoming the standard for multimedia data description (e.g. MPEG-7), and it is very suitable for Web-based data presentation owing to its expressiveness and flexibility. More specifically, video data can be separated into XML data description and raw videos. Using XML data description and a newly proposed Bi-Temporal Object-oriented XML Processing and Storage model, data integration and distribution can be easily achieved over the Web while the server keeps its own raw videos.

1.1 Background of Research

Over the last few years, we have been working on developing a generic video management and application processing (VideoMAP) framework [4], [5], [13]. A central component of VideoMAP is a query-based video retrieval mechanism called CAROL/ST, which supports spatio-temporal queries. While the original CAROL/ST has contributed on working with video semantic data based on an extended object oriented approach, little support has been provided to support video retrieval using visual features. To come up with a more effective video retrieval system, we have made extensions to the VideoMAP framework, and particularly the CAROL/ST mechanism to furnish a hybrid approach [6]. Meanwhile, we have worked out a preliminary version of our Web-based prototype called VideoMAP* [2], [3].

1.2 Paper Contribution and Organization

In this paper, we introduce a generic data conversion mechanism between Temporal Object-oriented Database and Temporal Object-oriented XML Data. The mechanism converts binary object data into XML packets for transmission. We also propose a Bi-Temporal Object-oriented XML Processing and Storage model as the central integration mechanism. The central mechanism can process and store infinite incoming XML packets (streams) from group of servers for data integration and distribution. The rest of the paper is organized as follows. We review some related work on Web-based video data model and issues of data streams in sect. 2. Sect. 3 is a preliminary introduction of our Web-based VideoMAP* framework. In sect. 4, we describe the architecture and mechanisms for data integration and distribution in detail. Finally, we conclude the paper and offer further research directions in sect. 5.

2 Related Work

2.1 Web-based Video Data Management

The DiVAN project [8] built a distributed audio-visual digital library system providing TV broadcasters and video archives owners with: (i) facilities to effectively compress, annotate, organize and store raw material; (ii) coherent content-based access facilities via user-friendly interface paradigms. It can demonstrate and assess the applicability and acceptability of the system through experiments based on the archives of major European broadcasters and audio-visual Institutions. The A4SM project [11] integrated an IT framework into video production process: (i) pre-production (e.g., script development, story boarding); (ii) production (e.g. collection of media-data by using an MPEG-2/7 camera); (iii) post-production (support of non-linear editing). In collaboration with TV-reporters, cameramen and editors, a MPEG-7 camera is designed in combination with a mobile annotation device for reporter, and a mobile editing suite suitable for generation of news-clips.

2.2 Data Stream Modeling

Data streams become useful for online analysis of continuous data e.g., sales transaction, stock market. Compared with the large volume of incoming data streams, most systems for stream processing have relatively limited amount of memory and space. Whatever the size of data streams, data streams will be removed once they are processed. Important information extracted from the data streams is stored in a data summary (intermediate results) repository for further processing. For example, sales transactions need to be stored in customers' profiles. Therefore, some ACID (atomic, consistent, isolation, and durability) characteristics of a relational database are necessary to maintain and some data models should be derived in order to handle data streams over the Web. Dionisio et al. [7] proposed a unified data model that represented multimedia, timeline, and simulation data utilizing a single set of related data modeling constructs. They developed a knowledge-based multimedia medical distributed database, which can handle time-based multimedia streams with structural information. Another research effort advocates a Data Stream Management System (DSMS) for network traffic management [1]. DSMS supports collecting and processing data streams using four data stores (Stream, Store, Scratch, and Throw). In addition, another complete DSMS called STREAM (Stanford stREam datA Manager) [9] is to be built to manage continuous and rapid data streams, which will have functionality and performance similar to a traditional DBMS.

2.3 XML Data Stream

For the purposes of data integration and distribution, XML (text-based) format is suitable due to its expressiveness and presentation power over the Web. Ives et al. [10] provided an overview of their Tukwila data integration system, which is based on a query engine designed for processing network-bound XML data sources. Tufte et al. [16] proposed merge operation and template for aggregation and accumulation of XML data. XML document can be created by integrating several XML documents, which are collected from the Internet. Their approach is simply to merge the documents semantically together, but without updating. Besides the research work on XML data integration, recent research on punctuated XML streams focuses on the format of XML streams, which aim is to handle fast processing of sequence of streams [14].

Suppose some blocking operators (such as sort) from data streams may block and delay stream processing, data streams should be embedded with some priori knowledge, which can be expressed in the form of punctuations.

3 Overview of the Web-based Object Video Management System

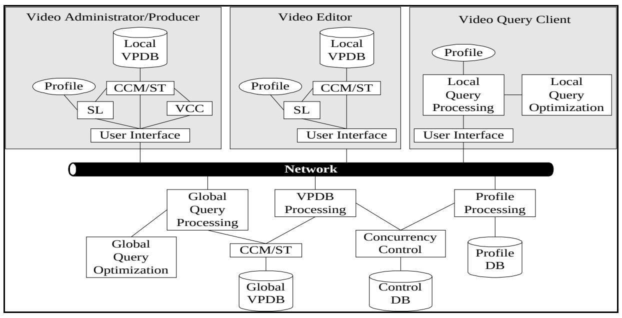


Fig. 1. Architecture of VideoMAP*: a Web-based Object Video Management System

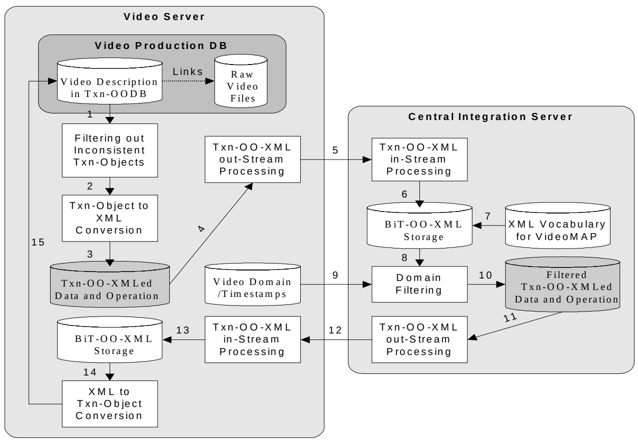
3.1 VideoMAP* Architecture

The architecture of our Web-based object video management system (VideoMAP*) is shown in Fig. 1. There are two types of components: server- and client- components. The client-components are grouped inside the gray boxes; the server-components are shown below the network backbone. The main server-components providing services to the clients include Video Production Database (VPDB) Processing, Profile Processing, and Global Query Processing. Four kinds of users can utilize and work with the system: Video Administrator (VA), Video Producer (VP), Video Editor (VE), and Video Query Client (VQ). Each type of users is assigned with a priority level for the VPDB Processing component to handle the processing requests from a queue. The user priority levels of our system, in descending order, are from VA, VP, VE, to VQ. Detailed discussions of the main components are described in [2].

3.2 Global and Local Video Production Databases

In the Web-based environment, cost of data communication is an important issue worth great attention. In order to reduce this overhead, databases can be fragmented and distributed into different sites. There are many research efforts on data fragmentation of relational databases. On the contrary, there is only limited recent work on data (object) fragmentation in object databases [12], [15]. In VideoMAP*, different types of users can retrieve the Global Video Production Database (Global VPDB) into their Local VPDBs for processing, and their work may affect, and be propagated to, the Global VPDB. In particular, each type of users may work on a set of video segments and create dynamic objects out of the video segments. Therefore, a generic mechanism for object data integration and distribution is necessary to address the problem of inconsistency among databases.

4 Object Data Integration and Distribution



Video Server
Video Production DB
Video Description in Txn-OODB
Links
Raw Video Files
1
Filtering out Inconsistent Txn-Objects
2
Txn-Object to XML Conversion
3
Txn-OO-XMLed Data and Operation
4
Txn-OO-XML out-Stream Processing
5
15
Video Domain /Timestamps
BiT-OO-XML Storage
13
Txn-OO-XML in-Stream Processing
14
XML to Txn-Object Conversion
Central Integration Server
Txn-OO-XML in-Stream Processing
6
BiT-OO-XML Storage
7
XML Vocabulary for VideoMAP
8
9
Domain Filtering
10
Filtered Txn-OO-XMLed Data and Operation
11
12
Txn-OO-XML out-Stream Processing

Fig. 2. Add-on Components for Object Data Integration and Distribution

4.1 Architecture and Process Flow

In Fig. 2, there are two servers: Video Server (VS) and Central Integration Server (CIS). In VS, there are four data stores: (i) Video Production Database (the VPDB described in sect. 3); (ii) temporary Transaction-timestamped Object-oriented XML Database (Txn-OO-XMLed Data and Operation); (iii) Bi-Temporal Object-oriented XML Database (BiT-OO-XML Storage); (iv) Video Server Domain Database (Video Domain/Timestamps). Besides, processes and mechanisms in between those data stores include (i) filtering process of inconsistent transaction-timestamped objects (Filtering out Inconsistent Txn-Objects); (ii) conversion processes between transaction-timestamped objects and XML (Txn-Object to XML Conversion, and XML to Txn-Object Conversion); (iii) transaction-timestamped object-oriented XML stream processing (Txn-OO-XML in-Stream/out-Stream Processing). In CIS, there are three data stores: (i) BiT-OO-XML Storage; (ii) Filtered Txn-OO-XMLed Data and Operation; (iii) XML Vocabulary for VideoMAP. The processes in CIS include Txn-OO-XML in-Stream/out-Stream Processing, and Domain Filtering.

The numberings shown in Fig. 2 depict the process flow of data integration and distribution between VS and CIS. Starting from process 1 of VS, some inconsistent objects (cf. Fig. 3) from the Transaction-timestamped Object-oriented Database (Txn-OODB) are first filtered out by the algorithm (i.e., *FilterInconsistentObject*) described in Appendix. Then, follow process 2 to 3, inconsistent objects are converted to XML format with proper transaction timestamps (Txn-OO-XMLed). The outputs of the filtering and conversion processes are shown in Fig. 4, Fig. 5, and Fig. 6 respectively. The description of each component and process is provided in the subsequent sections. The Txn-OO-XMLed Data and Operation component in VS is a temporary data store, in which XML data will then be trimmed into small pieces for sending out to CIS. XML data received by CIS are processed by the structural information of VideoMAP (XML vocabulary for VideoMAP), and the output is stored in the Bi-Temporal Object-oriented XML (BiT-OO-XML) Storage finally. Data from various VSs are integrated into CIS. According to the domain of each VS, data is then filtered and distributed to VS (from process 9 to 13). Once groups of XML packets are collected in the BiT-OO-XML Storage in VS, the data will be converted instantly to Txn-OODB (from process 14 to 15) for VPDB processing.

4.2 Component Description

Video Production Database. Video Production Database (VPDB) is a temporal (transaction-timestamped) object-oriented database designed for object data integration and distribution. With the introduction of transaction timestamps, inconsistent objects can be easily filtered out. Meanwhile, with the introduction of duplicate (backup) data, object can be rollback if the process of integration and distribution in CIS failed. It is also necessary to identify equivalent and/or identical objects in CIS, in which the objects are logically the same but exist separately among various VSs. Txn-OODB includes some base classes: *TxnObject*, *ObjectId*, *ObjectRef*, and *TxnTimestamp*. All classes of VideoMAP can be derived from *TxnObject*, which is embedded with three main features of object orientation: object identifier, object inheritance, and object composition. ObjectId contains a global identifier (*global_id*), a local identifier (*local_id*), and a *version*. With a *global_id*, object can be identified among different sites. *local_id* is for local processing in local site. *version* is useful for object versioning. *TxnObject* only stores the object identifiers of other objects for inheritance relationships and composite relationships. Therefore, conversion and extraction of binary objects (*TxnObject*) are relatively easy to perform, and dynamic construction of video programs can be performed. The structure of *ObjectRef* consists of an update frequency and a timestamp, type of operation, and a reference counter. All these data items are valid in the current local site only. The timestamps are obtained from a centralized controller once the user logons to the system. This information may be useful for integration.

Inconsistent Object Filtering. Inconsistent objects in Txn-OODB can be filtered out using the algorithm (i.e., *FilterInconsistentObject*) in Appendix. The timestamp *TS* of the last updated VPDB by CIS is used to compare with the transaction timestamps stored in each object. If one of the timestamps in an object is greater than *TS*, it means that this object is in an inconsistent state, and the object should be filtered out. An example of two inconsistent objects is shown in Fig. 3. The object shown in the left-side box was previously inserted with a timestamp of “20010102.223411”. The format of the timestamp is “yyyymmdd.hhmmss”. Then, data integration and distribution was performed at the timestamp of “20010102.223411”, therefore *TS* was set to “20010102.223411”. At the timestamp of “20020907.0000”, the “name” attribute of the left-object was updated. At this moment, a second object (shown in the right-side box of Fig. 3) is being inserted into the database with the timestamp of “20020908.105502”. As the second object is a child object of the first one, the inheritance relationship between these two objects is being updated at this time, with the timestamp of “20020908.105502”. Therefore, by comparing *TS* with the timestamps of objects, these two objects are marked as inconsistent.

```
object Scene : TxnObject
  attributes
    id.global_id.SITE="192.168.1.100"
    id.global_id.SEQ=1
    id.local_id.SITE="192.168.1.100"
    id.local_id.SEQ=1
    id.version=1

    freq_info.update_freq=2
    freq_info.update_freq_ts.last_update="20020908.105502"
    freq_info.operation_type="update"
    freq_info.reference_count=1

    // static data
    name="MyFirstScene"
    name_bak="MyScene"
    name_ts.last_update="20020907.000000"

    // dynamic inheritance
    parent_id_list=0
    parent_id_list_bak=0
    parent_id_list_ts.last_update="20010102.223411"
    child_id_list[0].global_id.SITE="192.168.1.100"
    child_id_list[0].global_id.SEQ=2
    child_id_list[0].local_id.SITE="192.168.1.100"
    child_id_list[0].local_id.SEQ=2
    child_id_list[0].version=1
    child_id_list_bak=0
    child_id_list_ts.last_update="20020908.105502"

    // dynamic composite
    component_id_list=0
    component_id_list_bak=0
    component_id_list_ts.last_update="20010102.223411"
    container_id_list=0
    container_id_list_bak=0
    container_id_list_ts.last_update="20010102.223411"
```

```
object Scene : TxnObject
  attributes
    id.global_id.SITE="192.168.1.100"
    id.global_id.SEQ=2
    id.local_id.SITE="192.168.1.100"
    id.local_id.SEQ=2
    id.version=1

    freq_info.update_freq=1
    freq_info.update_freq_ts.last_update="20020908.105502"
    freq_info.operation_type="insert"
    freq_info.reference_count=1

    // static data
    name="MySecondScene"
    name_bak=""
    name_ts.last_update="20020908.105502"

    // dynamic inheritance
    parent_id_list[0].global_id.SITE="192.168.1.100"
    parent_id_list[0].global_id.SEQ=1
    parent_id_list[0].local_id.SITE="192.168.1.100"
    parent_id_list[0].local_id.SEQ=1
    parent_id_list[0].version=1
    parent_id_list_bak=0
    parent_id_list_ts.last_update="20020908.105502"
    child_id_list=0
    child_id_list_bak=0
    child_id_list_ts.last_update="20020908.105502"

    // dynamic composite
    component_id_list=0
    component_id_list_bak=0
    component_id_list_ts.last_update="20020908.105502"
    container_id_list=0
    container_id_list_bak=0
    container_id_list_ts.last_update="20020908.105502"
```

Fig. 3. Example of Inconsistent Objects

```
<Operation type="update">
   <Object type="Scene">
   <ObjectId>
     <id.global_id.site value="192.168.1.100" />
     <id.global_id.seq value="1" />
     <id.local_id.site value="192.168.1.100" />
     <id.local_id.seq value="1" />
     <id.version value="1" />
   </ObjectId>
   <StaticData>
     <name value="MyFirstScene" />
     <name_bak value="MyScene" />
     <name_ts.last_update value="20020907.000000" />
   </StaticData>
   <Inheritance>
     <child_id_list.0.global_id.site value="192.168.1.100" />
     <child_id_list.0.global_id.seq value="2" />
     <child_id_list.0.local_id.site value="192.168.1.100" />
     <child_id_list.0.local_id.seq value="2" />
     <child_id_list.0.version value="1" />
     <child_id_list_bak value="0" />
     <child_id_list_ts.last_update value="20020908.105502" />
   </Inheritance>
  </Object>
</Operation>
```

```
<Operation type="insert">
   <Object type="Scene">
   <ObjectId>
     <id.global_id.site value="192.168.1.100" />
     <id.global_id.seq value="2" />
     <id.local_id.site value="192.168.1.100" />
     <id.local_id.seq value="2" />
     <id.version value="1" />
   </ObjectId>
   <StaticData>
     <name value="MySecondScene" />
     <name_bak value="" />
     <name_ts.last_update value="20020908.105502" />
   </StaticData>
   <Inheritance>
     <parent_id_list.0.global_id.site value="192.168.1.100" />
     <parent_id_list.0.global_id.seq value="1" />
     <parent_id_list.0.local_id.site value="192.168.1.100" />
     <parent_id_list.0.local_id.seq value="1" />
     <parent_id_list.0.version value="1" />
     <parent_id_list_bak value="0" />
     <parent_id_list_ts.last_update value="20020908.105502" />
   </Inheritance>
  </Object>
</Operation>
```

Fig. 4. Example of Inconsistent Object to XML Conversion

XML Data Conversion from Inconsistent Object. After filtering out the inconsistent objects, they are converted into plain-text XML format. The XML format would restore the object-oriented features and contain transaction timestamps. It also includes operation information. Such information is important and useful for other databases to do further processing. Then each attribute (such as name) of Scene is compared with its backup value (i.e., name_bak). If the timestamp of the attribute is greater than TS (cf. *FilterInconsistentObject* in Appendix) and the value is different from its backup value, this attribute will be copied to the format as shown in Fig. 4. The format of the result is named Transaction-timestamped Object-oriented XML Data (Txn-OO-XMLed Data and Operation).

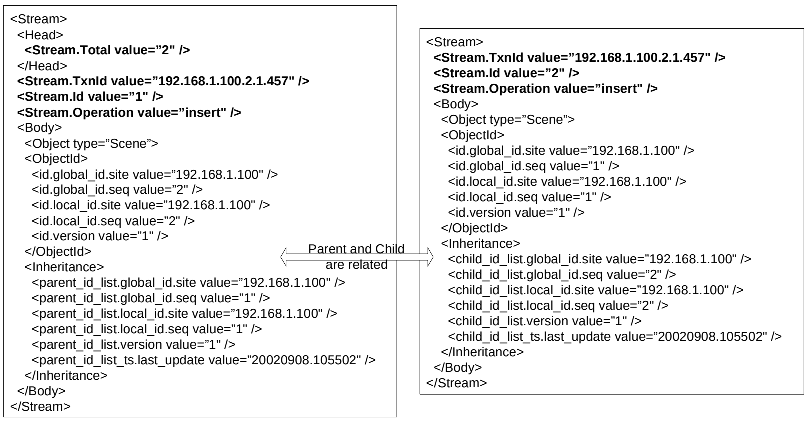
```
<Stream>
 <Head>
  <Stream.Total value=”1" />
 </Head>
 <Stream.TxnId value=”192.168.1.100.1.1.345" />
 <Stream.Id value=”1"/>
 <Stream.Operation value=”update" />
 <Body>
  <Object type=”Scene”>
  <ObjectId>
   <id.global_id.site value=”192.168.1.100" />
   <id.global_id.seq value=”1" />
   <id.local_id.site value=”192.168.1.100" />
   <id.local_id.seq value=”1" />
   <id.version value=”1" />
  </ObjectId>
  <StaticData>
   <name value=”MyFirstScene” />
   <name_ts.last_update value=”20020907.000000" />
  </StaticData>
 </Body>
</Stream>
```

```
<Stream>
 <Head>
  <Stream.Total value=”1" />
 </Head>
 <Stream.TxnId value=”192.168.1.100.2.1.456" />
 <Stream.Id value=”1" />
 <Stream.Operation value=”insert" />
 <Body>
  <Object type=”Scene”>
  <ObjectId>
   <id.global_id.site value=”192.168.1.100" />
   <id.global_id.seq value=”2" />
   <id.local_id.site value=”192.168.1.100" />
   <id.local_id.seq value=”2" />
   <id.version value=”1" />
  </ObjectId>
  <StaticData>
   <name value=”MySecondScene” />
   <name_ts.last_update value=”20020908.105502" />
  </StaticData>
 </Body>
</Stream>
```

Fig. 5. Example of Independent XML Packets

XML Processing. In order to allow immediate (stream) processing in CIS (due to limited size of space and memory of buffers), two groups of XML data as shown in Fig. 4 are further trimmed into four smaller pieces (i.e., XML packets shown in Fig. 5 and Fig. 6) in VS. If the object data is a simple attribute type (static data), the inconsistent value (e.g. name) will be copied to the packet. However, if the data is a list (e.g. parent_id_list), then the list is needed to compare with its backup value in order to further trim the elements of the list into more pieces (cf. *ObjectList2Packets* in Appendix). Two out of the four packets (cf. Fig. 7) can be processed independently (immediately), but some cannot.[1] Therefore, dependent packets are grouped together for transmission. The first packet of each group contains the head of packet ("/Stream/Head"). Inside the Head, it stores the total number of data packages ("/Stream/Head/Stream.Total") in each group. If this value is one, it means the packet is an independent packet. Each group of packets has a unique transaction identifier ("/Stream/Stream.TxnId"). Each packet has a packet transmission order ("/Stream/Stream.Id") and the type of operation ("/Stream/Stream.Operation"). There are three types of operations: update, insert, and delete. More specifically, operations are: (i) insert a new object; (ii) delete an object; (iii) insert object id to the object id list; (iv) delete object id from the object id list; (v) update static attribute data. Fig. 5 shows two independent packets, each belonging to its own group; Fig. 6 shows two dependent packets from an identical group.

[1] Therefore, data processing in CIS is semi-streaming.



```
<Stream>
  <Head>
    <Stream.Total value="2" />
  </Head>
  <Stream.TxnId value="192.168.1.100.2.1.457" />
  <Stream.Id value="1" />
  <Stream.Operation value="insert" />
  <Body>
    <Object type="Scene">
    <ObjectId>
      <id.global_id.site value="192.168.1.100" />
      <id.global_id.seq value="2" />
      <id.local_id.site value="192.168.1.100" />
      <id.local_id.seq value="2" />
      <id.version value="1" />
    </ObjectId>
    <Inheritance>
      <parent_id_list.global_id.site value="192.168.1.100" />
      <parent_id_list.global_id.seq value="1" />
      <parent_id_list.local_id.site value="192.168.1.100" />
      <parent_id_list.local_id.seq value="1" />
      <parent_id_list.version value="1" />
      <parent_id_list_ts.last_update value="20020908.105502" />
    </Inheritance>
  </Body>
</Stream>
```

```
<Stream>
  <Stream.TxnId value="192.168.1.100.2.1.457" />
  <Stream.Id value="2" />
  <Stream.Operation value="insert" />
  <Body>
    <Object type="Scene">
    <ObjectId>
      <id.global_id.site value="192.168.1.100" />
      <id.global_id.seq value="1" />
      <id.local_id.site value="192.168.1.100" />
      <id.local_id.seq value="1" />
      <id.version value="1" />
    </ObjectId>
    <Inheritance>
      <child_id_list.global_id.site value="192.168.1.100" />
      <child_id_list.global_id.seq value="2" />
      <child_id_list.local_id.site value="192.168.1.100" />
      <child_id_list.local_id.seq value="2" />
      <child_id_list.version value="1" />
      <child_id_list_ts.last_update value="20020908.105502" />
    </Inheritance>
  </Body>
</Stream>
```

Parent and Child are related

Fig. 6. Example of Dependent XML Packets

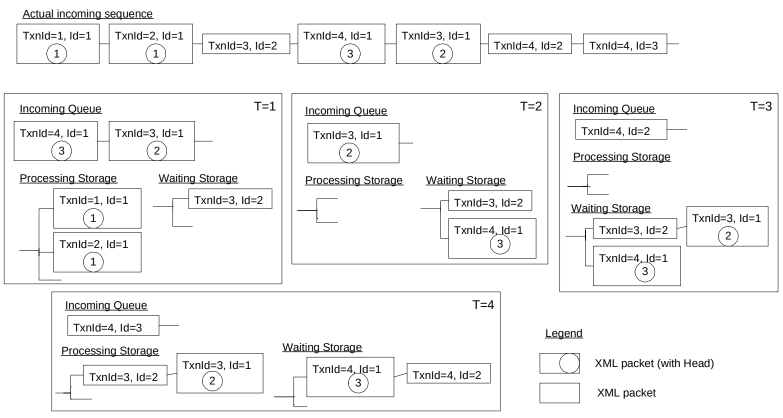


Fig. 7. XML Packet Processing

A scenario of XML packet transmission is shown in Fig. 7. XML packets are sent from VS to CIS during the process of Txn-OO-XML out-Stream Processing. The first two packets (i.e., with TxnId=1 and TxnId=2) are independent packets of Fig. 5. The dependent packets of Fig. 6 are illustrated by the packets with TxnId=3. The Actual Incoming Sequence shows the packets received by CIS. CIS receives the packets and stores them in the Incoming Queue. In addition, two temporary storages are used to process the incoming packets. As memory may be limited, the window processing size for the Processing Storage can be set to an optimal value according to the configuration of CIS. Besides window processing size, there are some other parameters and checking needed (e.g., time period for processing, checking of whether the size of next packet is over the size of window, and checking of the size of packet, etc.). For simplicity of the scenario, the value of the window processing size is infinite here, and all packets in the Processing Storage can be processed in one unit time of timestamp. Fig. 7 shows four snapshots of the process from timestamp T=1 to T=4. At T=1, the first two independent packets are received and stored into the Processing Storage for instant processing. As the third packet is a dependent packet, it is received and stored into the Waiting Storage. At T=2, the packet of TxnId=4 is stored into the Waiting Storage as it should wait for its dependent packets. Then at T=3, all packets in the group of TxnId=3 are completely received, therefore at T=4, this group of packets is sent to the Processing Storage.

The Bi-Temporal Object-oriented XML Model. Packets from the Processing Storage of Fig. 7 are stored into a Bi-Temporal Object-oriented XML (BiT-OO-XML) model. The model is bi-temporal because it contains transaction timestamps of the incoming XML packets and valid timestamps. By the assumption of limited space of CIS and large amount of video descriptions from VPDBs, data stored in the model will be removed periodically and/or when it is demanded, according to the valid timestamps. Therefore, the temporal structure of the model is designed for easy removal of data (cf. Fig. 8). According to the amount of resources, the structure grouped in the darker grey box is created and defined dynamically by the server with valid time. The root node named as “3d” means the model stores data for three days only. (It can be five days or even longer). The nodes in the second level (or further levels) further divide and group the packets according to the valid time.

Objects inside XML packets are to be stored in the structure below the grey box (cf. Fig. 8). Objects from the same transaction are first grouped together. Then the objects are clustered by their object types (e.g. scene, video), and are stored in a Data Pack with indices and a new valid timestamp. Once a Data Pack is formed, a scheduler would trigger the distribution process from CIS to all VSs. When the valid timestamp is expired, the Data Pack will be removed. Otherwise, if an object inside the Data Pack needed to be updated (by *global_id*, *local_id*, *version*, and *txn_timestamp*), the updated object will be migrated to a new Data Pack with a new valid timestamp and the distribution process will be triggered. We assume that objects are evolved from the same source, i.e. they are of their unique global ids. Different users can get an object, modify it, and then store it with a new *local_id*. However, *global_id* will never change. Other logical index structures can be formed (e.g., domain and any important attribute), with links to the physical structure. As domain information can be collected from VS, objects created from VSs with similar domains can be grouped together in the index structure. In order to facilitate the retrieval of XML data, indexes are created according to the XML Vocabulary for VideoMAP.

Domain Filtering. Domain filtering is to extract necessary data from Bi-Temporal Object-oriented XML Storage of CIS by the requirement specified by the VS. An indexing structure shown in Fig. 8 has grouped objects together from different VSs, which have similar domains information. Moreover, data can also be extracted based on the timestamps. Therefore, packets from different VSs with similar domain can be integrated and sent back to VSs in order to complete the cycle of data integration and distribution. VPDB can then be updated to a consistent state or started the process of rollback.

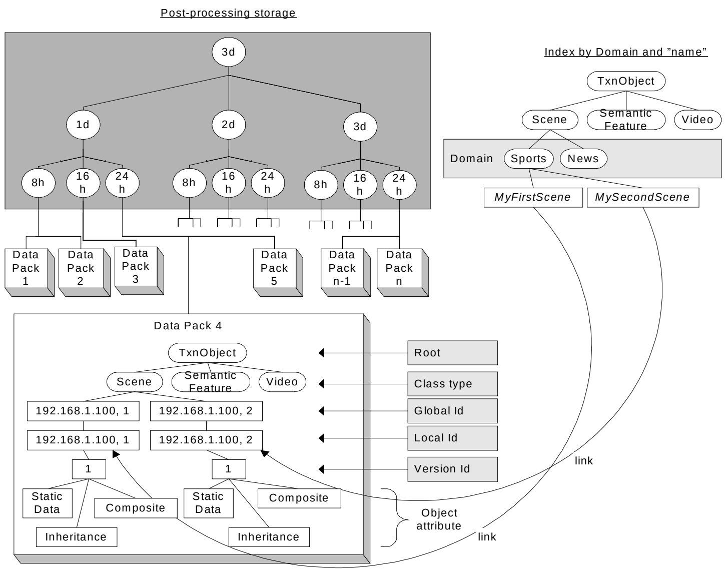


Fig. 8. Bi-Temporal Object-oriented XML Packet Storage

5 Conclusion and Future Research

In this paper, we gave an overview of our Web-based object video management system (i.e., VideoMAP*), from which the problem of data (object) inconsistency arose. In order to fix the problem, we introduced a generic data conversion mechanism between Temporal Object-oriented Database and Temporal Object-oriented XML Data. Both temporal data include transaction-based timestamps. In addition, we proposed a Bi-Temporal Object-oriented XML Processing and Storage model as the central integration mechanism. There are a number of issues for us to further work on. Among others, we plan to develop an *object XML* query language with a sufficient expressive power to accommodate VideoMAP* user queries adequately. Performance evaluation of processing such object XML queries is another important issue and will be conducted upon the VideoMAP* prototype system.

References

1. Babu, S., Subramanian, L., Widom, J.: A Data Stream Management System for Network Traffic Management. Proc. Workshop on Network-Related Data Management (2001)
2. Chan, S.S.M., Li, Q.: Architecture and Mechanisms of a Web-based Video Data Management System. Proc. IEEE ICME, New York City (2000)
3. Chan, S.S.M., Li, Q.: Architecture and Mechanisms of a Multi-paradigm Video Querying System over the Web. Proc. Int. Workshop on Cooperative Internet Computing, Hong Kong (2002) 17-23

4. Chan, S.S.M., Li, Q.: Facilitating Spatio-Temporal Operations in a Versatile Video Database System. Proc. Int. Workshop on Multimedia Information Systems (1999) 56-63
5. Chan, S.S.M., Li, Q.: Developing an Object-Oriented Video Database System with Spatio-Temporal Reasoning Capabilities. Proc. Int. Conf. on Conceptual Modeling (1999) 47-61
6. Chan, S.S.M., Wu, Y., Li, Q., Zhuang, Y.: A Hybrid Approach to Video Retrieval in a Generic Video Management and Application Processing Framework. Proc. IEEE ICME, Japan (2001)
7. Dionisio, J.D.N., Cárdenas, A.F.: A Unified Data Model for Representing Multimedia, Timeline, and Simulation Data. IEEE TKDE, Vol. 10, No. 5 (1998) 746-767
8. The DiVAN project. http://divan.intranet.gr
9. The DSMS project. http://www-db.stanford.edu/stream
10. Ives, Z.G., Halevy, A.Y., Weld, D.S.: Integrating Network-Bound XML Data. Bulletin of IEEE Computer Society TCDE, Vol. 24, No. 2 (2001) 20-26
11. The A4SM project. GMD IPSI – Integrated Publication and Information Systems Institute, Darmstadt. http://www.darmstadt.gmd.de/IPSI/movie.html
12. Karlapalem, K., Li, Q.: A Framework for Class Partitioning in Object Oriented Databases. Distributed and Parallel Databases, Kluwer Academic Publishers (2000) 317-350
13. Lau, R.W.H., Li, Q., Si, A.: VideoMAP: a Generic Framework for Video Management and Application Processing. Proc. Int. Conf. on System Sciences: Minitrack on New Trends in Multimedia Systems, IEEE Computer Society Press (2000)
14. The Punctuated Stream project. http://www.cse.ogi.edu/~ptucker/PStream
15. Ozsu, M.T., Valduriez, P.: Principles of Distributed Database System, Prentice Hall, Upper Saddle River, New Jersey (1999)
16. Tufte, K., Maier, D.: Aggregation and Accumulation of XML Data. Bulletin of IEEE Computer Society TCDE, Vol. 24, No. 2 (2001) 34-39

Appendix: Algorithms

```
FilterInconsistentObject
  input :
    objects of Video Production DB, IN-OBJ
    timestamp of consistent Video Production DB, TS
 output :
    inconsistent objects, OUT-OBJ
  begin
    /* Compare Transaction Timestamps */
    for each object A in IN-OBJ do
      if A.freq_info.update_freq_ts.last_update > TS, then
        Mark object A as OUT-OBJ
      else if there is any timestamp in object A > TS, then
        Mark object A as OUT-OBJ
      end-if
    end-for
  end. {FilterInconsistentObject}
```

```
ObjectList2Packets
  input :
    ordered object list, NEW-LIST
    ordered object list backup, OLD-LIST
 output :
    packets
 begin
    for each element A in NEW-LIST do
      compare A and with all elements in OLD-LIST
      if A is in OLD-LIST then
        Mark the element of OLD-LIST
      else if A is not in OLD-LIST then
        create an "insert" packet for element A
      end-if
    end-for

    for all unmarked elements in OLD-LIST
      create a "delete" packet for each unmarked element
    end-for
 end. {ObjectList2Packets}
```