Design and Representation of Complex Objects in Database Systems

Lin Qi, Huiyuan Zhang and Markus Schneider
University of Florida
Department of Computer & Information Science & Engineering
Gainesville, Florida, USA
{lqi, huiyuan, mschneid}@cise.ufl.edu

ABSTRACT

In recent decades, applications like geospatial, genomic, and multimedia have made use of very large and diverse application objects such as spatial networks and protein structures. These objects are complex in the sense that they are highly structured and of variable size. Storing, accessing and manipulating them in a standard and efficient manner is very challenging. The state-of-the-art solutions handle such objects by using file system formats like *HDF* and *XML*, serialization technique like *Protocol Buffers* and *BLOB* data type in databases. However, specialized file format solutions lack any well established database system features, and neither a uniform concept nor mechanisms exist for supporting complex objects for BLOBs. In this article, a novel and database-friendly framework of specifying and interpreting complex objects is proposed. Empirical studies have shown that our approach outperforms prevailing methods with efficient processing time and less storage consumption.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Applications – *data mining, spatial databases and GIS*

General Terms

Theory, Design, Performance

Keywords

Complex object, database system, big data

1. INTRODUCTION

The problem of handling large, variable-length and highly structured complex objects can be found in a wide range of applications such as spatial computing and spatial data analytics. Examples of such objects include biological sequence data, genomic data, multimedia data, imaging data and geospatial data. The demand of logical representation, efficient updating, fast transmitting, and space-saving features are increasingly important for complex applications, especially in a database context.

Traditional database management systems (DBMS) are well suited to store and manage large, unstructured alphanumeric data. However, storing and manipulating large, structured application objects at the low byte level as well as providing operations on them are hardly supported [2]. Specific file systems such as *Extensible Markup Language (XML)* [4] and *Hierarchical Data Format (HDF)* [6] define a set of rules for encoding documents. Those documents are in textual or binary data format, which is human-readable or machine-readable, are very suitable for describing objects with complex logical structure. However, due to the nature of markup languages, the contents of an object is not encapsulated in the file but rather stored in textual format, leading to a large size which is not ideal for reading or writing. Distributed file systems such as Hadoop [13] focus on data distribution, mapping and reduction, and do not provide explicit support for complex object management. On the other hand, serialization techniques like *Protocol Buffers* [3] enjoys efficient transmission and data manipulation. However, they are not intended for a smooth integration into database systems. A widely accepted approach of handling complex data in databases is to model and implement them as values of abstract data types (ADT) in a type system, or algebra, which is then embedded into an extensible DBMS and its query language [2].

In order to address the above problems, we propose a novel and generic method called *Complex Object Specification (COS)* in this article. COS is a flexible, efficient and automated mechanism for specifying and processing highly structured data. The design goals for COS emphasize agility and performance. It includes two parts: first, a specification language that provides grammars and rules to describe structured objects; second, it also includes a parser that generates source code from the various sources based on corresponding structural interpretations. COS provides an interface to describe the structure of complex objects at the conceptual level.

The remaining parts of this article are organized as follows. Section 2 describes relevant research related to complex object data storage and management. The COS framework is introduced in detail in Section 3. Section 4 presents empirical results of our proposed approach compared with three other widely used methods. Finally, Section 5 concludes the paper and discuss potential working direction.

Copyright 2015 ACM ISBN 978-1-4503-3967-4/15/11 $15.00
DOI: http://dx.doi.org/10.1145/2820783.2820842.

Table 1: Comparison of Different Approaches on Handling and Managing Complex Objects

Criteria	File System	Tables	BLOBs	Standalone	OO Extension	COS
1. Abstraction					✓	✓
2. Generality						✓
3. Proprietary			✓			✓
4. Domain-specific operation definition				✓	✓	✓
5. Query support		✓	✓	✓	✓	✓
6. Extendibility	✓				✓	✓
7. Efficient random access				✓		✓

2. RELATED WORK

Complex objects storage and management is an important research topic over recent years [1, 12, 11, 10]. Researchers are seeking opportunities and possible ways to handle complex application objects in DBMS to better manage and manipulate data in an organized and efficient manner. A data model born in database environment can also enjoy the advantages provided by database systems like concurrency controls, transaction processing, querying optimization, and recovery. The most prevailing and natural way of storing complex structured data is to make use of tables and BLOBs in traditional DBMS. Any hierarchical structure within an object can be incorporated in tables using a separate attributed column that maintains cross-referenced tuples with their primary keys. However, the main drawback of this method is that the overall abstract concept of the object is lost, i.e., the internals of a single complex object are spread over several tables. This is known as *abstraction problem*. This may lead to the fact that expensive joins are required to bring object information together.

Another trend in the database research stream is the development of new DBMS prototypes as standalone (or semi-standalone) data management solutions. These include systems such as SciDB [8], PostGIS [5], and BSSS [9]. Most of the systems offer low-level byte range operations for update and insertion and can handle variable-length byte sequences. However, these systems do not preserve or maintain structural information and are unable to provide dedicated and random access to sub-components.

Our proposed solution, the COS framework, takes in real application data and generate serialized yet formatted byte stream. It provides a generic specification of the complex structures of an object by using a specially designed description language. The specifications are then stored in a .cos file which will then be compiled into *accessor classes* and *utilization functions* that interpret the input data and perform serialization. The COS framework preserves the structural information of the object and can provide operation definitions from a high level perspective. It can also be extended to perform as the bridge between real data and DBMS. Table 1 provides a summary of existing approaches and out proposed solution across identified problems and important criteria.

3. COMPLEX OBJECT SPECIFICATION

Blocks and *properties* are typically used to describe complex structures. *Properties* are basic elements and have no further inner structure. Blocks can still be ”divided” into smaller structures including properties or lower-level blocks. A tree representation is a useful tool to describe hierarchical information. However, it is not machine friendly. A computer system cannot read the tree representation and construct the corresponding complex object easily as there is no standardized tree representation rules in both texts and patterns. In order to give a more precise description and to make it understandable to computers, a formal specification is required. Therefore, we propose a generic *Complex Object Specification Language (COSLang)* to describe the hierarchical structure of application objects.

Since properties and blocks are the two main basic elements to compose a highly structured object, we need an elegant way to describe them as well as preserve their structural information simultaneously. In COSLang, the definition of a *property* involves five parts shown in Figure 1 in order to capture enough information represented by a single property.

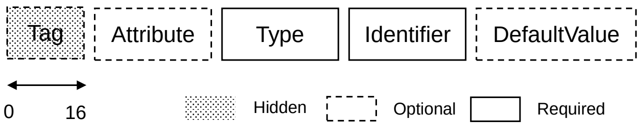

Figure 1: Format of property in COSLang

Tags are automatically assigned to each property and are used to identify different properties of data in serialized format of data. The field *Attribute* is an important prefix for a certain property. It is used to describe the nature, or certain features, of a specific property. There are five pre-defined attributes: *list*, *optional*, *unique*, *description*, and *obsolete*. *Type* is a required field in the property definition. COSLang provides five basic built-in types: *int*, *double*, *string*, *bool* and *bytestr*. In addition to these scalar value types, developers are allowed to create *user defined blocks (UDBs)* to register new types. *Identifier* serves as the ”name” of the property. *DefaultValue* assigns a default value to the property if no specific data is fed. This is an optional field.

An example of a property definition is as follows:

```
// Definition of property X_Corr and Name in COSLang
[description("X coordinate")] double X_Corr = 0.0;
[optional][description("Point name")] string name;
```

Next, we will introduce the format of a *block* definition in COSLang. As shown in Figure 2, each block definition comes with a block attribute, an identifier as well as a ”body” part. The hierarchical structure of the block is represented by the components inside the body definition. Generally, a simple block (as in Figure 2(1)) consists of a series of properties. For example, a *Point* type is composed by three coordinates in the 3D plane and each coordinate is a property of the point. Furthermore, there are blocks (as in Figure 2(2)) that consist of more complex structures with nested blocks defined on its deeper structure level. For instance, a *Segment* is a spatial line segment connecting two points. Note that each point object is also a block.

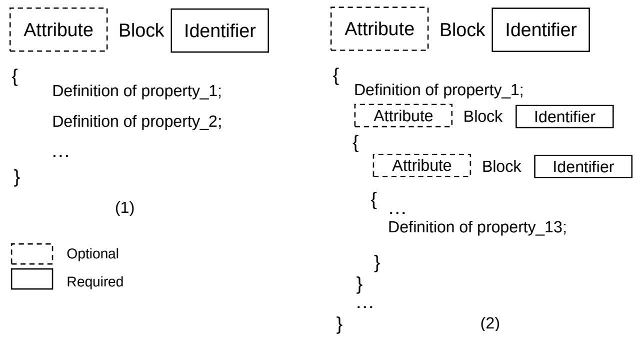

Figure 2: Format of (1) simple block (2) nested block in COSLang

3.1 Complex Object Specification Parser

In the previous section, we have introduced the basic concepts and schema of COSLang. However, being a specification language, it is not capable of handling and manipulating the input stream. In order to perform as the bridge between the programming language and the real data, a *Complex Object Specification Parser (COSParser)* has been devised to help object designers.

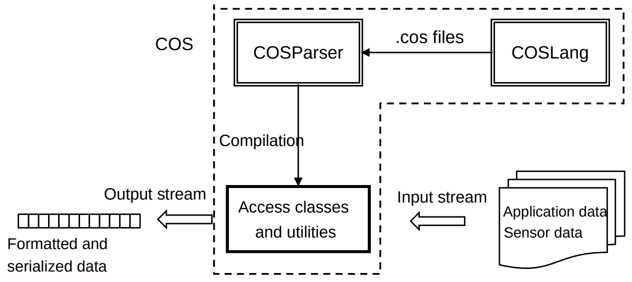

Figure 3: COS Architecture

The architecture of COS is shown in Figure 3. The role of COSParser is to parse the logical structure defined in *.cos* files based on COSLang and generate access classes as well as utility functions in programming languages such as C++ and Java etc.

The overall COS platform provides many advantages over file systems such as XML and HDF in the following ways:

- Light: specification language and COSParser are separated from real data, thus the framework itself works as a plugin toolkit;
- extensible: new properties or blocks can be added to .cos files without breaking the old code and application program, also called *backwards-compatibility*;
- Consistent: it is not allowed to edit the code generated from the parser. Any change needs to be performed in .cos files and then reflects to data access classes through compilation, which ensures the consistency between the specification file and the code.
- Flexible: most of the formats are self-describing; attributes provide a powerful yet clean way of imposing restrictions.

4. PERFORMANCE EVALUATION

In this section, we evaluate our COS framework performance by comparing it with three other file systems for handling structured objects. Currently we are actively implementing the database integration parts. From our experimental results, the COS framework can achieve more efficient running times for object creation and random read operations. It also consumes less storage compared with other prevailing approaches.

Experimental Setup. Among the freely available network maps data, OpenStreetMap (OSM) [7] is a comprehensive collection of road network data. We will be using the nodes inside those networks to generate one complex application object for evaluation.

In our testing datasets, the file sizes representing the networks range from 14 MB to 2.5 GB with 52,251 nodes and 12,003,566 nodes respectively. We are going to compare our COS framework with Text, XML and HDF file systems w.r.t. (1) object data creation time, (2) size of structured data stream, (3) read and write performance. In addition, we will also include the time spent on generating corresponding API code based on specifications in .cos file in our object creation. The testing environment is the Windows 8 system with Intel i7-4770 3.4 GHz CPU and 16 GB RAM with stable working state.

The four networks, from small to large, are Gainesville city, Alachua county, Orlando city and Florida state road networks shown in Table 2.

Table 2: Number of Nodes in Network Datasets

Gainesville	Alachua County	Orlando	Florida State
52,251	979,605	1,272,803	12,003,566

The *Text* method is a plain approach that writes data into textual documents with basic point indexes and no compression.

The first set of experiments is to measure the generation time of object data of different algorithms based on the dataset. In this experiment, we model all points within the point set as properties of one complex object. The generation times are shown in Table 3 in seconds. The time spent on loading the datasets is excluded.

From the results we can see that with the increment of the dataset, the time spent on generating object data is also increasing. The Text approach enjoys a steady increasing in generation time with high positive correlation based on data size. The trend for XML approach is similar but with much more time. As for the COS framework, it achieves the best results among the four approaches across all datasets. As mentioned in Section 3, the structure of the complex

Table 3: Generation Time of Object Data

Time (seconds)	Text	XML	HDF5	COS
Gainesville	0.046	0.676	0.775	0.009
Alachua County	0.783	9.307	0.801	0.138
Orlando	1.311	16.177	0.813	0.216
Florida State	11.698	157.833	0.995	0.907

object is pre-defined in a .cos file so that the generation step becomes easier and light. By loading the incoming data, the complex object is incrementally built in an efficient manner. In addition, the concept of data chunks is a perfect match with our streamlining strategy.

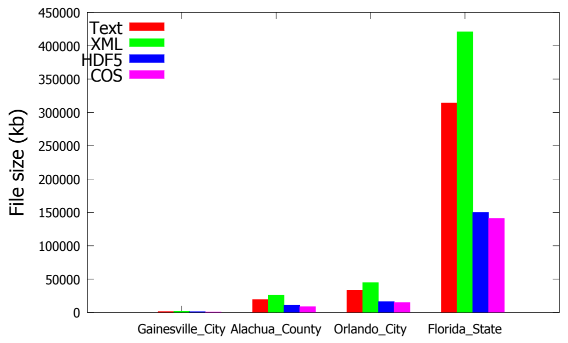

Figure 4: Storage Size Comparison

The next experiment is to evaluate the file size that contains the complex object. The size of files determines the efficiency of data transmission in a local environment as well as in distributed systems. In our experiment, file sizes are measured in the unit of *kb (kilo-bytes)* shown in Figure 4.

Random read is an important operation as the applications might be frequently asking for data for different components of the complex object. This operation is performed 10,000 times for each round of testing. The captured time is taken as the average time and measured in milliseconds. The results are shown in Figure 5.

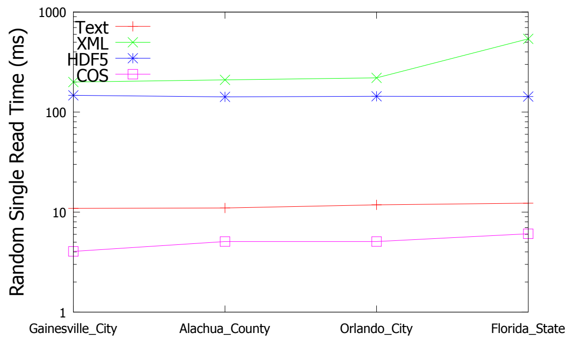

Figure 5: Random Read Time Comparison

The XML approach spent the largest amount of time on random read. As for the COS framework, the random read operation is extremely efficient due to that the structural indexing information, i.e., a tag map, is stored inside the headers of each page of data. Given the data request, COS can quickly locate the pages that contain this data, and then provide direct access to the data inside those pages based on the tag map information. The time spent on random reading from all datasets is around 5 ms and even for the largest dataset, it takes less than 10 ms.

5. CONCLUSION AND FUTURE WORK

In this paper, we provide a novel solution to store and manage complex application objects by proposing a new mechanism for handling structured objects inside DBMSs. The design goals for COS emphasize agility and performance.

The future work will include a careful implementation of the integration of the COS framework into DBMS. This needs to be done in a general sense so that COS can be applied independently of different DBMSs. In addition, the current COS framework processes the object data sequentially. The potential parallel computation techniques should be designed for complex object data management to further lift the efficiency.

6. REFERENCES

[1] Li-Ju Chen, Ying C Guo, Xin S Mao, Bo Yang, and Hua Zhang. Managing a complex object in a cloud environment, June 3 2014. US Patent App. 14/294,908.

[2] T. Chen, A. Khan, M. Schneider, and G. Viswanathan. iblob: Complex object management in databases through intelligent binary large objects. In *3rd Int. Conf. on Objects and Databases (ICOODB)*, pages 85–99, 2010.

[3] http://en.wikipedia.org/wiki/Protocol_Buffers. Wikipedia - protocol buffer.

[4] http://en.wikipedia.org/wiki/XML. Wikipedia - xml.

[5] http://postgis.net. Postgis.

[6] https://www.hdfgroup.org. Hdf group.

[7] http://www.openstreetmap.org. Openstreetmap.

[8] http://www.paradigm4.com. Scidb.

[9] B. Hwang, I. Jung, and S. Moon. Efficient storage management for large dynamic objects. In *System Architecture and Integration 20th EUROMICRO Conference*, pages 37–44. 1994.

[10] Lin Qi and Markus Schneider. Monet: Modeling and querying moving objects in spatial networks. In *3rd ACM SIGSPATIAL Int. Workshop on GeoStreaming (IWGS)*, pages 48–57, 2012.

[11] Markus Schneider, Shen-Shyang Ho, Malvika Agrawal, Tao Chen, Hechen Liu, and Ganesh Viswanathan. A moving objects database infrastructure for hurricane research: Data integration and complex object management. In *Proceedings of the Earth Science Technology Forum*, 2011.

[12] Alex Simpkins and Emanuel Todorov. Complex object manipulation with hierarchical optimal control. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 338–345. IEEE, 2011.

[13] Tom White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.