A Data Cube Model for Analysis of High Volumes of Ambient Data

Hao Gui and Mark Roantree

March 25, 2013

1 Introduction

The concept of the smart city [9] of which there are many initiatives, projects and demonstrators, is generally underpinned by one or more ambient systems parts that require a mediation process to deliver the interconnectedness required by an ambient system. Due to the high volumes of information involved in smart city and many ambient systems, it is inefficient to integrate ambient sources without first filtering data not relevant to immediate information needs. As part of a smart city initiative, we have developed a component called CityBikes which monitors availability of bicycles and parking slots in bicycle rental schemes run by the city [6]. Across twelve cities, this component generates 40-50 MBytes of data daily. Our motivation is to provide a method of efficient indexing and accessing data before merging the appropriate subsets into a large ambient information system. The challenge is that these types of ambient systems generate XML data, ideal for interoperability, but very slow for query processing and information extraction.

Contribution. We adopted a traditional data cube approach in order to aggregate and transform data for fast query processing. However, the majority of these approaches are dealing with relational and not XML data. The contribution in this paper is in the development a new framework for constructing and managing XML data cubes. As this framework is based on the high performing dwarf approach, we eliminate prefix and suffix redundancy to ensured condensed data cubes. As part of this framework, our XML Data Cube (XDC) model provides a purely XML solution to the process of analyzing and extracting XML streams of ambient data. Our evaluation focused on the applicability of our XML data cubes (in terms of memory size and speed on cube construction) and as part of that evaluation, we developed a synthetic data set and also tested using the real world city bikes dataset.

Paper Structure. The paper is structured as follows: in §2, we briefly describe the state of the art in XML data and online analytical processing using cubes; in §3, we review one of the stronger XML cube approaches as this provides the basis for our own work, and describe how we developed an XML approach; in §4, we present our metamodel for managing XML cubes and discuss analytical application in areas such as ambient systems; in §5, we present our evaluation; and finally in §6, we provide some conclusions. Due to space limitations, areas of this paper are kept deliberately brief.

2 Related Work

Reusing relational technology for XML data streams is problematic. The difficulty in converting between both formats strongly motivates the need for a purely XML approach as shown in surveys such as [3]. Here, the author provides a survey of the different approaches, detailing the open issues which are diverse enough to create a fundamental approach to providing a solution. This motivated our decision to begin with an XML cube metamodel which was sufficiently expressive to address the open issues.

In [1, 2], the authors are faced with a similar problem to that which is tackled here: data for the system is coming from the Web. Their approach is to create many attribute trees for the XML data and then to optimize the construction for the cube by using pruning and grafting functions. While their approach overlaps with ours in that both present a multidimensional (meta)model for cube representation and construction, there is no concrete analysis of performance in their work. Similar to our work, they use a real world dataset for evaluation but we have a detailed set of experiments as part of our evaluation.

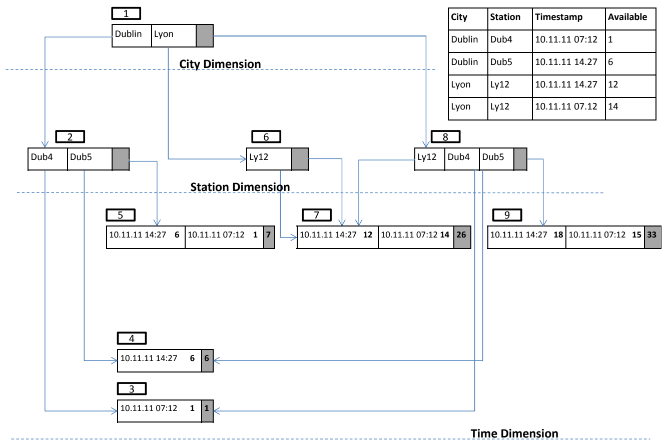
In [5], the authors tackle multidimensional XML and provide a means of updating using a graph model through a set of new functions. A new language MXPath is proposed with extensions over the standard XML language XPath. However, their approach does not employ the dwarf model and thus, will not benefit from the proven optimizations [4] of this approach. In fact, the work in [4] is the sole research project to use the full set of functionality we provide. However, their approach works only with relational data and cannot serve the many new data sources that provide XML data such as sensor networks and the sensor web.

3 Deriving an XML Data Cube Model

Both the physical and logical design for XML OLAP Data Cubes are very important for data analytics and mining tasks. The former contributes to the efficient storage, usage and aggregation of data where the criteria generally include the trade-off between minimum storage overhead with guaranteed data access, using well-designed data-structures or indexes. On the other hand, the logical design focuses on providing sufficient metadata and semantic information regarding the aggregation data in order to support many forms of OLAP operations. To a large extent, physical and logical designs drive the evaluation process and performance of the data cube.

By reviewing conventional OLAP technology over relational data, several physical models have been proposed (i.e. ROLAP, MOLAP, HOLAP). The differences between them are obvious, whether the aggregation data are stored in and managed by underlying database or some designated storage (multidimensional arrays for example). ROLAP has obvious advantages and can efficiently address all functionality in the life cycle of a data cube [7]. Our approach follows the basic concepts of ROLAP, that is to say, we want to develop an XOLAP (XML counterpart for ROLAP) model for data cube. However with ROLAP, one manages a data cube using a relational approach for underlying relational data, while the XOLAP approach aims to provide data cube management functionality using an XML model for XML data.

We began the development of our model by reusing the basic concepts in the well known Dwarf construction to derive a new XML Data Cube representation. In this work [10], they introduce the Dwarf structure and provide a detailed discussion on a sample structure similar to that used here. We have created an instance to model our requirements based on the CityBikes data streams. In brief: the sub-dwarf structure comprises a number of dwarfs; any dwarf D is defined as the node



City	Station	Timestamp	Available
Dublin	Dub4	10.11.11 07:12	1
Dublin	Dub5	10.11.11 14.27	6
Lyon	Ly12	10.11.11 14.27	12
Lyon	Ly12	10.11.11 07.12	14

Figure 1. The Dwarf Structure with datasets

D and all nodes which are connected by D's outgoing links; each of these dwarfs are said to be sub-dwarfs of D.

3.1 Dwarf Set and Tree Structure

The primary objective is to clarify the relationship between the Dwarf data structure and the schema of the XDC structure. Figure 1 illustrates what our CityBikes repository would resemble if modeled as a relational dwarf structure. To obtain a logical *tree* model from this Dwarf set, we make a number of observations. There are two types of node links with the first being the explicit link between a pair of labeled nodes, such as the link from the city **Dublin** to the station **Dub4**. Most links in the dwarf structure will be of this type. From the same illustration, we can see the gray node at the end of each dwarf set, referred to as an *ALL* cell. Its purpose is to aggregate all values of the node and was shown in [10] to provide a method for certain optimizations. The second type is the implicit link between any two successive nodes in a specific subdwarf. Thus, taking node 2 as an example, there exists an implicit link from **Dub4** to **Dub5**. Because every node can be regarded as a linked list separately, this explicit link is also directed, from the logical predecessor to successor, **Dublin** to **Dub4** in this case. We distinguish between the two types by saying that explicit links cross two adjacent dimensions while implicit links remain in a specific node.

Using dwarf logic, a simple transformation process can be used to map a set of subdwarfs (or dwarf nodes) to a tree structure, by firstly creating a binary tree, and then transforming into a regular tree using left-child right-sibling encoding. We now describe the transformation process informally in order to show the procedure intuitively.

The result of this process for the binary tree (a) and corresponding regular tree(b) are illustrated in Figure 2 (for simplicity, we use T1 and T2 for the 2 timestamps).

We have created a new property for the tree in Figure 2(b), denoted by the dashed line arrows used to capture redundant information in Dwarf construction. As a result, certain leaf nodes on the right-hand side tree are shown as semi-opaque, to indicate that they are candidates for reduction in our optimized tree.

The Dwarf construction employs a unique top-down computation strategy for the data cube,

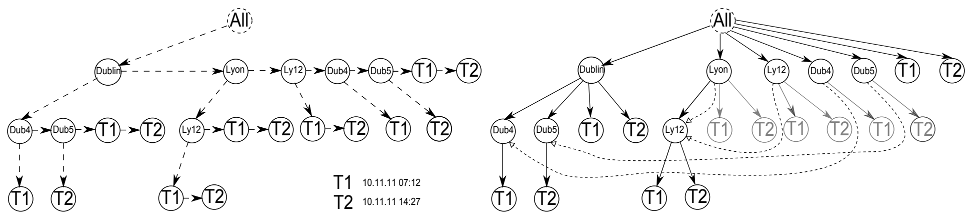


Figure 2. Tree structures derived from Dwarf Set: (a) binary tree (b) regular tree

which automatically discovers and eliminates all prefix redundancies on a given relational data set[10]. Prefix sharing is the basic design goal when using an XML hierarchical structure to represent complex data in a simple form. The benefit is a reduction of duplicates that occur in dwarf structures.

The motivation for transforming the common Dwarf structure into a semantically equivalent tree, which we refer to as a Dwarf Derived Tree (DDT), is that it provides a path to the development of an XML cube metamodel, necessary to eliminate redundancies and maintain condensed cubes. Furthermore, our Dwarf Derived Tree has the same density as the original Dwarf structure, already demonstrated to have excellent query response times [10] and elsewhere [4] shown to have other possibilities for optimization.

3.2 XML format representation for Data Cube

In figure 2, we provide the XML data cube for the DDT in Figure 2(b). This transformation is simple and preserves the fundamental characteristics of the dwarf structure, including both data cube descriptive characteristics and reduced density. As shown in figure 2, the node count and the aggregation value count in the XML data cube representation are identical to the dwarf structure in 1, providing an intuitive proof of indirect inheritance. With the elimination of "All" nodes, a large portion of tree nodes in DDT have been promoted to a higher tree level and are thus, more efficient to retrieve.

Furthermore, we deliberately employed some XML standards in our XML data cube representation, namely XLink and XPointer of W3C. The combination of these two can be used to implement pointers in Dwarf or DDT structures. For example in <C1 xlink:href='#3' xlink:role='out'>, xlink:href is an XLink, and its value '#3' is an XPointer expression. '#' notation is the addressing mechanism for id attributes in XML documents (of course the functionality of XLink and XPointer are much more powerful than this), and after storing this XML data cube into a native XML Database, the creation of ID Attribute Index (almost all native XML DBs support this kind of index) can improve the efficiency of the retrieval process significantly. Finally, we distinguish different types of XLink by using xlink:role, which could provide additional information when performing related OLAP operations.

4 XDC and Data Cube Computation

In section 3, we presented an XML version (in both tree and document formats) of the dwarf model. In this section, we present the XML Data Cube (XDC) metamodel together with a discussion on how

OLAP operations can exploit the metamodel for data mining purposes. The metamodel captures our framework for building and optimizing XML data cubes.

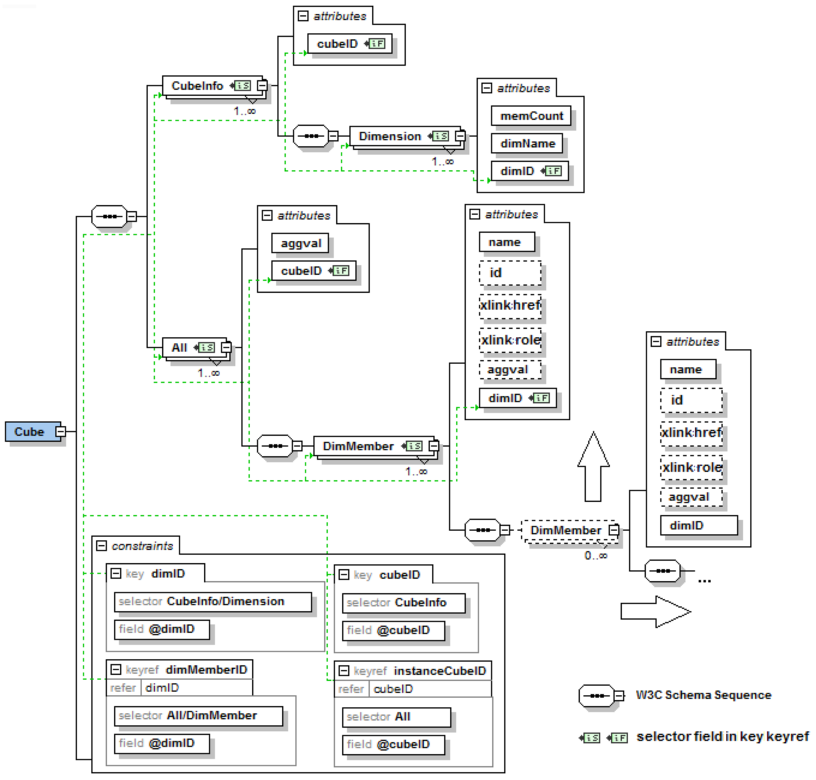


Figure 3. XDC Metamodel

4.1 XML Data Cube Metamodel

The XDC metamodel presented in figure 4 has a structure that provides sufficient information to carry out typical OLAP operations without the need to retrieve data from the underlying database. Furthermore, it minimizes the size of the cube itself by taking full advantage of the hierarchical nature of XML and its extensibility. There are three major components in the XDC metamodel which are described now.

- **Cubeinfo.** The CubeInfo element provides metadata concerning a cube, and dimension information such as name, id etc. While this is naturally independent of any specific cube instance, it will significantly improve data processing when maintained together with instance data. As the size of the metadata is relatively small, it incurs little storage overhead.
- **All.** The All element refers to the same element in our DDT in figure 2(b). It contains the complete instance specific aggregation data of all cuboids composing the current cube lattice. Attributes `id` and `aggval` are used to for identification and measurement data with respect to

each specific cuboid cell. If there is more than one aggregation value from measurement (such as sum and count), it is intuitive to turn `aggval` into a set of named attributes, even sub-elements, with structures to hold them. The key component here is the `DimMember` element which is self-contained and recursive from two directions, i.e. horizontally and vertically (explicitly marked). For horizontal nesting, according to figure 4, the child `DimMember` inside a parent `DimMember` can have any number of instances. In this way, `DimMember` can be used to capture the one to many logic inside the actual input data. The ellipsis notation in figure 4 illustrates vertical nesting: `DimMember` can be nested inside `DimMember` to arbitrary depth, and thus, used to represent the measurement data from a specific cuboid with specific series of dimensions.

- **constraints.** The dash lines with arrows represent constraints inside the data cube structure. They model the relationships between dimensional metadata and instance data using XML Schema key/keyref mechanisms to ensure the integrity of the entire cube.

4.2 Building XDC Cubes

The XDC construction process differs from that of Dwarf as the underlying data models differ. Furthermore, by examining and analyzing Dwarf construction in detail in previous work [11], we showed that prior to the construction, Dwarf construction requires that the fact table is sorted using a fixed dimension order. Dwarf construction requires this in order to carry out prefix expansion and suffix coalescing, the details of which are outside the scope of this paper. We present a more flexible construction mechanism that eliminates the prerequisite of sorting, as ambient and other sensor systems will involve streaming data that must be quickly captured with appropriate data cubes updated. As the XDC construction model focuses on creating and maintaining a DDT tree when processing input XML data, this can directly be serialized into a XDC instance, and essentially, we now have all of the beneficial properties of the dwarf model within the XDC framework.

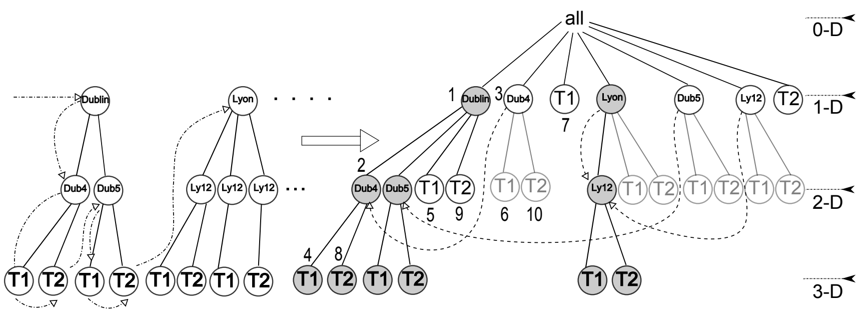


Figure 4. XDC construction

The corresponding DDT tree is shown on the right-hand side of figure 4 which contains gray nodes (hereafter referred as Data Nodes) and white nodes (hereafter referred as Link Nodes). Data Nodes form the basic skeleton frame and at the same time host cube data for several cubes (namely City, City-Station and City-Station-Time). Each Data Node in the DDT contains four types of metadata: parent-child links in the tree structure; redundancy link as a DDT tree derived from Dwarf; actual attribute values for related dimensions; and most importantly, the aggregation data for corresponding cube cells. A Link Node also contains the first three types of metadata but instead of the actual aggregation data, the most important information it contains is view links consisting of all the Data Nodes in its set. In other words, Data Nodes provide aggregation data for data cubes, while Link Nodes offer viewing angles from different aspect. Nodes in semi-opaque format in figure 4 are those that are eliminated as a result of suffix redundancy.

Assuming node ALL is at level 0, then any node at level *i* represents a cube cell from the corresponding *ith* dimension in the data cube. This tree structure can easily handle multidimensional data in the XML hierarchy, with a simple transformation to XML. We will now provide a simple example.

The dashed lines in figure 4 link the first XML elements in order (left-hand side input stream): Dublin->Dub4->T1->T2->Dub5->T1->T2->Lyon. In the DDT tree on the right, the creation order for those XML elements is shown. Each Data Node, regardless of its level, has a set of observer Link Nodes. For example, the observer Link Nodes for Dublin->Dub4->T1 include Dublin->T1, Dub4->T1 and T1, while Dublin->T1 is the observer of both Dublin->Dub4->T1 and Dublin->Dub5->T1. When the construction algorithm encounters any data items in the XML stream for the first time, then after creating the corresponding Data Node, it must also traverse the DDT to locate (or create) all its observer Link Nodes and build the relationships between them. At this point, the algorithm also checks for redundancy and if located, ignores the entire subtree for that node and creates a redundancy link. If any node violates the condition of suffix redundancy, the redundancy link is deleted and the view links used instead.

5 Performance and evaluation

In this section, we provide an evaluation of an XDC instance during the construction process. All the experiments were performed on an Intel Core2 E8400 PC clocked at 3.0GHz and with 4GB of memory. As the platform was 64-bit Windows7, it enabled the usage of the full 4GB of physical memory. Both original input XML data and the constructed XDC instance data are stored and retrieved from a native XML database – XHive. The default DOM and SAX parsers shipped with JDK1.7.0 are used, and other XML data handling tasks are accomplished by using Saxon9 open-source implementation version 9.3.0.5. The test dataset includes two categories, one is generated by our own software (for experiment1), the other from our ongoing analysis [6] of bicycle sharing schemes (for experiment2). SAX is used for cube construction; DOM for OLAP cube operations, with both being Saxon implementations.

5.1 Results from the Synthetic Data (Experiment 1)

In order to obtain knowledge of the applicability of XDC, we evaluated the construction of various cubes with the results shown in figure 5. A large volume of XML data was automatically generated for each text, and for the construction of each XDC instance, we recorded the performance and evaluating the average cost.

As shown in figure 5, the construction performances for different data in terms of number of data entries (100K, 500K), dimension cardinality (5,10,15,20,25), and number of dimensions (3,4,5,6,7) are shown. Unfortunately, we cannot directly compare the performance of dwarf construction [10] with that of XDC construction as a direct comparison is not possible: the underlying database systems are different, data models are different, and thus, data processing costs must differ. Specifically, XDC construction must parse the input XML data stream to extract each value from characters

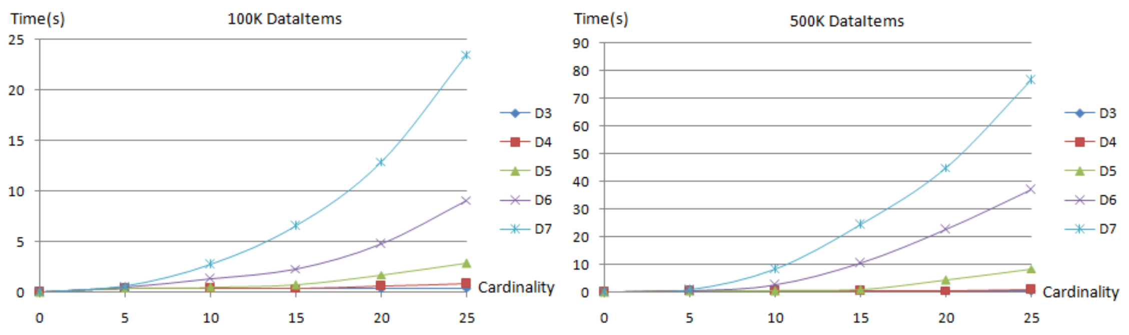


Figure 5. Experiment 1. Synthetic Dataset

for a specific dimension in an event driven style, which is more expensive than relational Dwarf. Furthermore, the dwarf approach requires sorting in advance, which effects an increase in efficiency. Given that our average cube size was roughly 100MB, our results are comparable with [10] in terms of acceptable times for cube construction, if we consider their results for 15 dimensions with a dwarf size of 153MB, with a build time of 68 seconds. While our approach is faster, we must allow for their older machines and smaller amounts of memory.

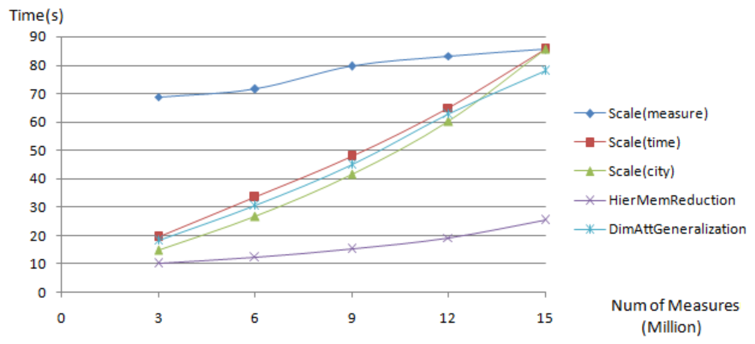


Figure 6. Experiment 2. Citybikes Dataset

5.2 Results from the Citybikes Data (Experiment 2)

For our second evaluation, our motivation was to use the real-world dataset that provides the focus for our work. It includes over 15,000,000 samples of bicycle usage (3GB in total) in XML format from a number of cites around the globe [6]. There are five dimensions and two concept hierarchies in this experiment and we elected to cut it from different angles. Our approach was as follows:

- In the legend in figure 6, for Scale (measure), we cut the original dataset by bike measurements, because the bikes usage sampling frequency is 30 times per hour, so cutting from this aspect will obviously reduce the size, without any significant effect on the structure of input data.
- In legend Scale (time) and Scale (city), we cut the original dataset by dimensions hour and city, and thus, it affected both size and structure in different ways, and the performance curves

followed the same increasing pattern when size of datasets increased for each parameter. Because three Scale parameters were used in the same dataset at value 15 on the X-axis, three curves converged at this point. In addition, reduction in dimension city resulted in comparatively large change in structure, so the performance of Scale(city) is a little bit better than that of Scale(time).

In addition, we carried out two other tests using this dataset. In the city dimension, there is a concept hierarchy consisting of city and station, where some cities have large numbers of stations (for example, Bruxelles has almost 180 stations). In the first test, we deleted one of the members in this hierarchy (namely station) to examine the impact on data cube construction. Because the cardinalities of the station members vary in size and are very large overall, there was a significant improvement in performance. This is illustrated in figure 6 by the HierMemReduction plot.

For the second test, we chose the weather condition dimension to implement generalization. We imposed a more strict discretion rule to reduce the weather condition into either Good or Bad rather than its original values (such as "Fair", "Thunder", "Partly Cloudy" and so on) [6], because we made the decision that some attributes such as the the specific kind of weather conditions (40 in total), do not make a significant contribution to the analysis of bicycle usage statistics. The result shown in figure 6, brought some performance improvements. However, it appears that the effects were far less than the reduction of the concept hierarchy member station. The reason is that the cardinality of the dimension weather condition is large but the actual value distribution of this dimension is very sparse, as the weather condition changes little, for a given city even over an entire day.

6 Conclusions

In many ambient systems, large volumes of data can be quickly created due to continuous monitoring using sensors and ready access to ambient data. While this presents a powerful new information source, the volumes of data and information overload can make information extraction a slow process. In this paper, we presented a new XML Cube Metamodel which is a native XML model that captures cube metadata without omission of data or redundancy. It bases itself on the original dwarf approach to deliver these optimizations while proving for multidimensional modeling of XML data. This addresses the problem of information overlap by providing a framework for constructing XML data cubes to both reduce information overload and to optimize extraction of data. Our current focus is on expanding our construction model to allow dynamic updates to the data cubes as data arrives in a real-time format and thus, keeping cubes current as the ambient system adapts to changes in the environment.

Acknowledgement.
This work is jointly supported by the School of Computing Research Committee Fund at Dublin City University and by Science Foundation Ireland under grant 07/CE/I1147.

References

[1] Boussaid O., Ben Messaoud R., Choquet R., and Anthoard S. X-Warehousing: An XML-Based Approach for Warehousing Complex Data. Proceedings of ADBIS, LNCScience vol. 4152 pp.39-54, 2006

[2] Boussaid, O., J. Darmont, F. Bentayeb and S. Loudcher. Warehousing complex data from the web. Int. J. Web Eng. Technol., 4: 408-433, 2008.

[3] Cuzzocrea A. Cubing Algorithms for XML Data. Proceedings of DEXA Workshops, IEEE Computer Society, pp. 407 - 411, 2009.

[4] Dittrich J., Blunschi L., and Salles M. Dwarfs in the Rearview Mirror: How Big are they Really? Proceedings of the VLDB Endowment, pp.1586-1597, 2008.

[5] Fousteris N and Gergatsoulis M. Updating Multidimensional XML Documents, Intl Journal of Web Information Systems 4(2), pp 142-164, 2008.

[6] Gerard Marks, Mark Roantree and Dominick Smyth. Optimizing Queries for Web Generated Sensor Data. 22nd Australasian Database Conference (ADC2011), pp. 151-159 Australia, 2011.

[7] Morfonios K. and Ioannidis Y. Supporting the data cube lifecycle: the power of ROLAP. VLDB Journal 17:4, pp.729-764, 2008.

[8] Rusu, I.L., R. Wenny and T. David. Partitioning methods for multi-version XML data warehouses. Distributed Parallel Databases, 25: 47-69, 2009.

[9] Schaffers et. al. Smart Cities and the Future Internet: Towards Cooperation Frameworks for Open Innovation. In Future Internet Assembly, LNCS vol. 6656, pp. 431-446, 2011.

[10] Sismanis Y., Deligiannakis A., Roussopoulos N., and Kotidis Y. Dwarf: Shrinking the PetaCube. Proceedings of ACM SIGMOD, pp 464-475, 2002.

[11] Gang Long Xiang, Yu Cai Feng, Hao Gui. Construction and compression of Dwarf. Journal of Zhejiang University SCIENCE, 2005 6A(6):519-527