# Extended Term tXSchema uses SAX Parsing Partially

Ratnaparkhi Punam S
Student of MEIT,
Amrutvahini College of Engineering, Sangmner

S.E.Pawar
Head of the Department, Information Technology
Amrutvahini College of Engineering, Sangmner

## ABSTRACT
World Wide Web Consortium recommendation for XML Schema illustrates the structure and data types of XML document. tXSchema could be a framework for the creation and validation of time variable documents. All parts of tXSchema changes over time to replicate changes reference world of information. tXSchema could be a model for making temporal Schema from base Schema, logical and Physical Annotations. We also describe how the validator can be extended (temporal validator) to validate documents in this seeming uncertain situation of data that vary across time while its corresponding schema and even its representation are also varies. Since many applications require to keep tracks of the Schema, data evaluation, which suggested a need of versioning. When system is working with huge versions of schema as well as xml data files here we are specially focusing on how to minimize processing time by using SAX parser's partial involvement with conventional DOM parser. In this paper we deal with versioning in tXSchema model, more precisely here we propose a set of Schema change primitive for the maintenance of logical and physical annotations and define their operational perspectives and also minimizes processing time.

## General Terms
XML Editors, Physical and Logical Data Independence, Temporal; XML Data.

## Keywords
XML Schema, Temporal Database, Physical Annotations, Logical Annotations, XML constraints, XML validation.

## 1. INTRODUCTION
XML is becoming a popular language for documents and data which published on the web. XML used to create any XML document designer need to refer what should be the structure of a document, but the structure of XML document is an XML Schema. Every XML document generated is always validated against its Schema which ensures confirmation to the formatting rules for an XML document, also confirmation to the types, attributes specify by the Schema itself. One of the main concerns of XML document is its time varying nature. An XML schema suggests the description or structure of the XML document; typically it would be presented in terms of constraints on the Schema and content of documents. A temporal document used to record the transition of a document over time, i.e., all of the past versions as well as the most current version of the XML document. These databases or document generally contains information about when things are happened. So in this paper we will test how to preserve and validate time-varying data within XML Schema. One approach would have been to propose changes to XML Schema to accommodate time-varying nature.

Temporal XML Schema (tXSchema) is responsible for constructing and validating temporal documents tXSchema could be a framework for the creation and validation of our time variable documents. All parts of tXSchema changes over time to replicate changes to reference world of information. Temporal Annotations makes the conventional XML Schema document become Temporal XML document .Physical annotation, describes how to represent the time-varying aspects of the document. A temporal bundle is nothing but, the XML document that serves as temporal Schemas together with the non-temporal schema, and temporal annotation and physical annotation. So that XML Schema is compatible with both XML Schema and the XML data model. We then extend tXSchema to support schema versioning. When the schema is versioned, the base schema and temporal and physical Schemas can themselves be time-varying documents, in doing so, we use both conventional XML Schema and related tools (importantly, the conventional validator), as well as VALIDATOR for data versioning.

A challenge with schema versioning is that anything can modify or update, and thus must be versioned it can be anything such as the snapshot documents, the base schema, the temporal annotations, the physical annotations, the schema documents [16]. With the framework introduced in a paper, we will show that we can:

1) Implementing XML Editor, for interactively working on the schema, schema versioning, and temporal document only.

2) Achieve logical data independence by specifying what can change in the temporal annotation.

3) Achieve physical data independence by specifying the location of timestamps in the physical annotation.

4) Here we are exploring SAX-based temporal constraint validation techniques.

### 1.1. Key Issues
a) While working temporal xml schema are as temporal term relates to changing of data. Traditionally real time systems manage their data in application dependent structure. As record keeping system evolve like inventory management systems, their applications become more complex and require access to more data. Applications such as these rely on temporal databases, which record time-referenced data. Which store only facts which are believed to be true at the current time, here we are focusing on amount of data being processed which is huge as it have facility of keeping historical files as well as method of roll backing.

b) In computing, a parser is a program (or a piece of code or API that you can reference inside your own programs) which analyses files to identify the component parts [17]. All applications that read input have a parser of some kind, otherwise they'd never be able to figure out what the information means [17]. In computing, a parser is a program (or a piece of code or API that you can reference inside your own programs) which analyses files to identify the component parts. All applications

never be able to figure out what the information means. XML applications are just the same: they contain a parser which reads XML and identifies the function of each the pieces of

the document, and it then makes that information available in memory to the rest of the program [17].

While reading an XML file, a parser checks the syntax, well-formedness, and reports any violations or errors. While parsing XML document with conventional parser, DOM, The Document Object Model parser is a hierarchy-based parser that creates an object model of the entire XML document, then hands that model for you to work with.

## 2. ISSUES WITH TRADITIONAL APPROACH

A schema language for a temporal document needs to have some way of specifying and enforcing such constraints. The conventional XML Schema validator is also incapable of a time-varying document using the representational schema. Actually, XML Schema is not sufficiently expressive to add temporal constraints.

For example, XML Schema cannot specify the following (desirable) schema constraint: the transaction-time lifetime of a <ID> element should always be contained in the transaction-time lifetime of its parent <emp> element. Second, a conventional XML Schema document added with timestamps to denote time-varying data cannot be used to validate a snapshot of a time-varying document. For example, XML Schema cannot specify the following (desirable) schema constraint: the transaction-time lifetime of a <ID> element should always be contained in the transaction-time lifetime of its parent <emp> element. Second, a conventional XML Schema document augmented with timestamps to denote time-varying data cannot, in general, it is used to validate a snapshot of a time-varying document.

A snapshot is an instance of a time-varying document at a single point in time. For instance, if the schema asserts that an element is mandatory (minOccurs=1) in the context of another element, there is no way to ensure that the element is in every snapshot since the elements timestamp may indicate that it has a shorter lifetime than its parent (resulting in times during which the element is not present, violating this integrity constraint); XML Schema provides no mechanism for reasoning about the timestamps. Even though the representational and snapshot schemas are closely related, there are no existing techniques to automatically derive a representational schema from a snapshot schema (or vice-versa).

The absence of a automatic technique implies that user need to depend on specially appointed strategies to develop a representational pattern. Depending on specially appointed techniques will confine information autonomy. The planner of a diagram for time-changing information needs to settle on a mixture of choices, for example, whether to timestamp with periods or with worldly components, which are sets of non-covering periods and which components ought to be time-differing. By receiving a layered methodology, where the depiction XML Schema, transient annotations, and physical annotations are particular archives, singular outline plan choices can be determined and changed, regularly without affecting the other configuration choices, or in reality, the processing of tools.

For example, a tool that computes a snapshot should be concerned primarily with the snapshot schema; the logical and physical aspects of time-varying information should only affect (perhaps) the efficiency of that tool, not its correctness. With physical data independence, few applications that are concerned with representational details would need to be changed. Hence, an improved tool support for representing and validating time-varying information is needed. Creating a time-varying XML document and representational schema for that document is potentially labor-intensive. Currently users has to manually edit the time-varying document to insert timestamps indicating when versions of XML data are valid (for valid time) or are present in the document (for transaction time).

The user also has to modify the snapshot schema to define the syntax and semantics of the timestamps. The entire process would be repeated if a new timestamp representation were desired. It would be ideal to have automated tools to create, keep up, and redesign time-shifting records when the representation of the time stamped components changes.

One challenge with schema versioning is that, in this potential quicksand, anything can change, and thus must be versioned: the snapshot documents, the base schema, the temporal annotations [16], the physical annotations, the schema documents included by these documents, even the schemas of these schema components [3].

And, as a result of the physical annotations can change, the concrete representation within a temporal XML document will vary. Thus, it becomes even more difficult to even outline validation in such a fluid atmosphere. Schema versioning should offer a solution to the above problem by enabling intelligent handling of any temporal mismatch between data and its schemas. A framework is required that might retain past information and past schemas, whereas permitting the present information and schema to be extracted.

Processing tons of data within small period is one more key issue, previous approaches by Chomsky [1] present DOM parser for processing such temporal database, which can be very efficient for particular number of data files, if we process real time databases let's say e commerce websites then it definitely need the optimize solution for processing huge database which can give smaller time value.

This work has several real-world applications. As an example, the Botanic Garden and Botanical Museum in Berlin-Dahlem (BGBM1 ) maintains a repository of XML Schemas2 related to index terms, keywords, biodiversity data about specimens and observations, meta-level information regarding collections, organizations,

and networks, and various wrapper and configuration files. Most of those XML schemas have had multiple versions over the last 2 to 3 years.

## 3. PROPOSED SYSTEM

### 3.1 Proposed Approach

SAX parser: An event-based sequential access parser API that only operates on portions of the XML document at any one time. SAX parsers have some benefits over DOM-style parsers. A SAX parser only needs to report each parsing events it happens, and normally discards almost all of that information once reported. Thus, the minimum memory needed for a SAX program is proportional to the utmost depth of the XML file (i.e., of the XML tree) and also the most knowledge concerned in an exceedingly single XML event (such as the name and attributes of a single start-tag, or the content of a processing instruction, etc.)

A DOM computer program, in distinction, generally builds a tree illustration of the whole document in memory to start with, therefore using memory that will increase with the whole document length. This takes wide time and area for big

documents (memory allocation and data-structure construction take time). The compensating advantage, of course, is that when loaded any a part of the document are often accessed in any order.

Because of the event-driven nature of SAX, process documents are mostly way quicker than DOM-style parsers, ciao because the process may be wiped out a start-to-end pass. Many tasks, like compartmentalization, conversion to different formats, very simple formatting, and the like, can be done that way. Other tasks, such as sorting, rearranging sections, getting from a link to its target, wanting up info on one part to assist method a later one, and therefore the like, need accessing the document structure in advanced orders and can be a lot of quicker with DOM than with multiple SAX passes.

## 3.2 Plan of work: XML partial processing with SAX:

A parser that implements SAX (i.e., a SAX Parser) functions as a stream parser, with an event-driven API. The user defines a number of callback methods that will be called when events occur during parsing. SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again. But DOM Parser is rich in functionality because it creates a DOM tree in memory and allows you to access any part of the document repeatedly and allows you to modify the DOM tree. A SAX Parser, however, is much more space efficient in case of big input document (because it creates no internal structure).

Our proposed system basically intent to reduce the processing timing, Following are the relevant procedural steps required to do so:

1) Constraint adding, & validating

2) Squashing

3) Schema changing or generating temporal document along with temporal schema and temporal xml data files.

4) Unsquashing

5) Validation (Tool: TempValidtor).

For satisfying our aim we use SAX approach partially, wherever reading of xml document considered. What SAX does is to read XML document sequentially, it loaded one data slice at a time and process where DOM loaded entire XML file. For this purpose we have generated SAX reading code residing in SAX package separately and export this package to file name representationFactory.java. This file is actually generated to process the input and send it to called procedures like squash, unsquash and tempValidator. So when squash get called from user it first parse all the input files via representationFactory module, if input files are missing say temporal schema it displays error. Same with unsquash and tempValidator modules. so here we are using both parser same time, when reading of particular input we used SAX technique of parsing and when modifying particular xml file we are using DOM parsing , partial use of SAX parsing technique gives processing time minimal as compared to fully DOM parsing system, only if we are working on huge dataset.

# 4. SYSTEM ARCHITECTURE

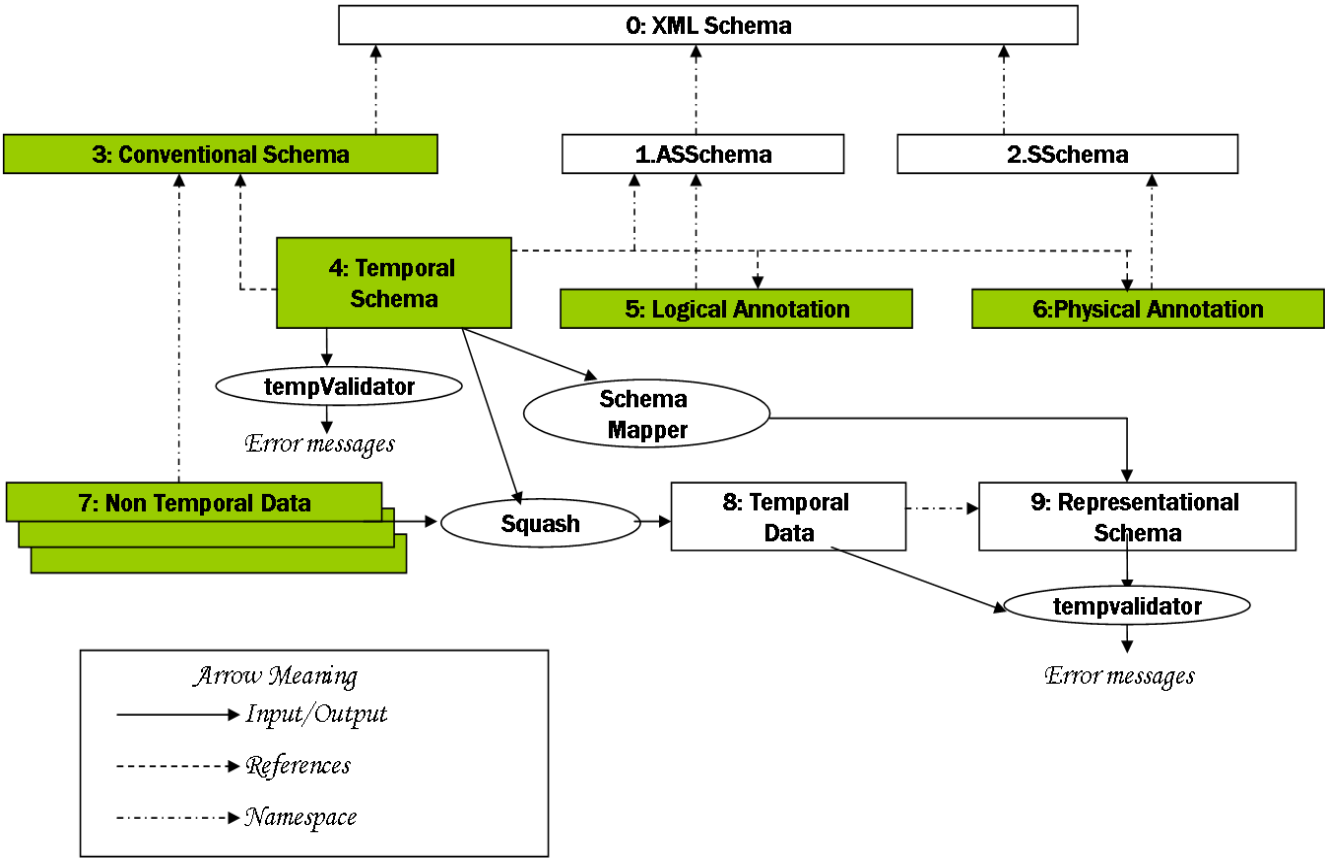


**Figure 1: System Architecture**

When changes occur in base Schema, we annotate base Schema with the logical annotations; attributes of logical annotation can be valid time, transaction time, continuous state, single event, etc. When logical annotations are written on the Base Schema, whole document featured as a Logical Schema. We can observe that few of the component from figure are highlighted that are specific to an individual time-varying document and should be supplied by the user.

Non temporal content or static content which will not change across the time and are kept constant throughout the different versions of base Schema, shown in box7. After that, physical annotation on conventional Schema is done by time stamping techniques, shown in box 6.

Which is chosen by the user, such as where the timestamps are placed and their kind (e.g., valid time or transaction time) and the type of representation adopted. Two documents with the same logical annotation look different if we change or replace the location of their physical timestamp.

Annotation Schema (ASchema) is annotation document that represents Schema document with both logical and physical annotation. Where TSSchema document is a combination of a conventional schema document as well as the logical annotation .Finally, the designers tie up conventional schema, logical annotation, and physical annotations together that forms standard XML document.

Validator tempvalidator will be processed temporal schema document in order to ensure that the logical and physical annotations are

(1) Valid with respect to Annotation Schema, and

(2) Consistent with the conventional Schema. Tempvalidator generated reports (such as error messages) about valid and invalid temporal Schema document. SCHEMA MAPPER generates the output as a representational Schema by validating consistency of annotation presented in temporal Schema, Sequence of non-temporal documents and temporal Schema can be squashed into temporal document. Moreover, tempValidator is used to validate temporal data against the representational schema through that reports whether the temporal data document is valid or invalid. Our editor can also do the reverse process of Squashing called as Unsquash, which will able to convert representational Schema into document slices.

## 4.1. Steps of working System

Conventional schema and XML data files(Non temporal Data) this data files are nothing but dataset on which we are working is of company. Company database includes details of employee, supplier, and product where the related rules are included in schema. Each XML document will relate to company schema needs to validates all constraints specify by the schema.

### *4.1.1. Temporal Schema*

Step 1:

In this project we found that XML versioning can cause by two reasons:
1) When constraints get changed or need to change. It directly causes changes into xml schema. We can consider it as major change because many XML document needs to validate with the corresponding new schema. In this project these changes are done by using annotation file to specify the changes into the schema.

2) When there is need to modify the previous schema validated xml document, we can modify it n save it also compile it, but we can also modify the data and store it as a new version of document and compile it as a new document.

Step 2:
Changing content itself makes data temporal. TSSchema i.e. temporal schema document imports xml schema of company as well as annotation document. TDSchema i.e. temporal xml data document file consist of tracking of all versioned XML documents in the form of slices. ASchema contains logical and physical annotation file that can available when designer wants to change constraints. This indirectly changes schema. The new file introduce here because it allows changing the original schema without actually making physical changes into the original schema, it simply reflects the changes logically.

### *4.1.2. System Modules*

**SAX parsing Steps**:
In this project we use SAX for reading purpose only when there is requirement of read and write at the same time such as creating versions then we use DOM parser. We create package SAX read where we store the code of parsing with SAX following are the steps which used while reading input documents:

SAX Parsing Method:

1) Creating the Skeleton

2) Importing Classes

3) Setting Up I/O

4) Implementing the ContentHandler Interface.

5) Handling Content Events

6) Setting up the Parser

7) Setting up Error Handling

8) Implementing SAX Validation

9) Running the SAX Parser Examples with Validation

**Squash:**
The SQUASH utility takes a sequence of XML documents, a temporal annotation and a physical annotation as input and generates a temporal XML document consistent with the physical annotation. The algorithm for SQUASH tool is given in Figure It cleverly reuses pushUp, pushDown and coalesce primitives to create a compressed document from a set of snapshot documents as per the given temporal schema

The Squash first checks for the consistency of the temporal and the physical annotations with the snapshot schema. It then creates a new XML document with <timeVaryingRoot> as its root and attaches root elements of the snapshot documents as its versions. At this point, the timestamps are present at the root level element. pushDown function then moves these timestamps down the hierarchy to the elements present in the temporal annotation. Every item is then coalesced to create its compact representation. The pushUp function then moves the timestamps up in the hierarchy up to the elements present in the actual physical annotation.

**Unsquash:**
The Unsquash utility performs the opposite operation of SQUASH. It takes a temporal XML document, a temporal bundle and generates multiple non-temporal XML documents. It also provides the functionality of extracting a particular

snapshot from the given temporal document using UNSQUASH utility.

The Unsquash first checks for the consistency of the temporal and physical annotations with the snap-shot schema. It then constructs the representational schema using SCHEMA MAPPER and parses the given temporal document against the representational schema using the conventional validator. The pushdown function is first called on the given document to move the timestamps to the temporal elements. A new physical annotation, containing only the root element, is created and passed to the function pushUp. The purpose is to move all the timestamps to the root element. At this moment every version of the root item element is a snapshot document. These individual versions are then written to the separate files.

**Temporal Validator:**
TempValidator provides the validation procedure. The temporal bundle document is passed through the tempValidator which first checks to ensure that the temporal and physical annotations are consistent with the snapshot schema and with each other. Once the annotations are found to be consistent, the logical-to-representational Mapper (SCHEMA MAPPER) generates the representational schema from the original snapshot schema and the temporal and physical annotations. The representational schema is needed to serve as the schema for a time-varying document and is used to validate the temporal document using conventional validator.

## 5. RESULT ANALYSIS AND EVALUATION

We now study the performance of squash , unsquash and tempvalidator when we uses SAX to parse the input files for all three. For showing the time difference between conventional method and proposed method of validation we execute validation process and record the values.

Unsquash is specifically used while adding new versions or changing schema, and when there is need to view documents separately, we didn't include it in analysis purpose of time instead we use it while working on different data files to satisfy other relevant goals. TempValidator is the validating tool for proposed system which accepts SAX parsing input where XMLLint is base paper validating tool which follows DOM approach throughout[1].

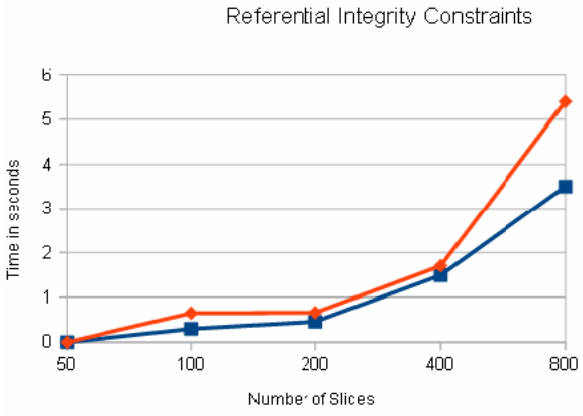


**Figure 2: Validating time required for Referential Integrity Constraints by both the tools.**

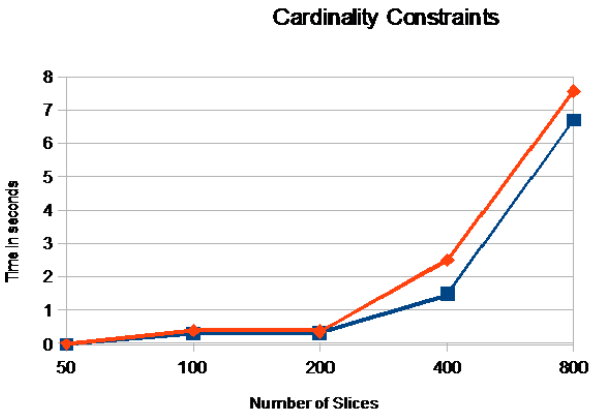


**Figure 3: Validating time required for Cardinality Constraints by both the tools.**

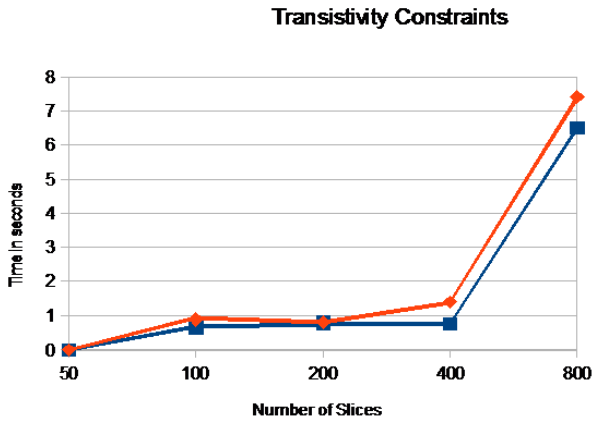


**Figure 4: Validating time required for Transitivity Constraints by both the tools.**

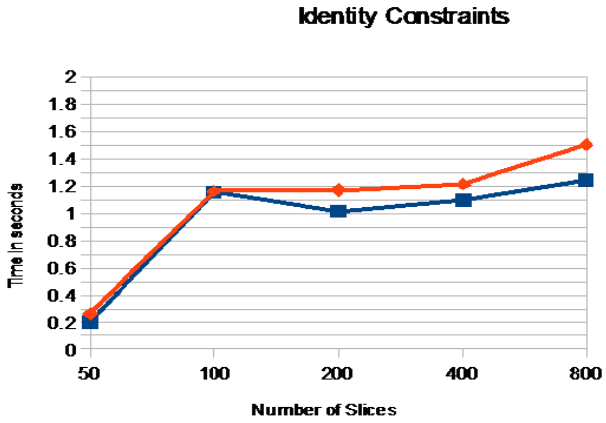


**Figure 5: Validating time required for Identity Constraints by both the tools.**

We are focusing on how the time required to process the document goes from steps squashing to Validating with partial support of SAX for input processing and with conventional parser support i.e. DOM throughout the processing. To do so we use a dataset of company, that supports versioning. Data files varies from 50 to 800, so the versions are created. For taking more no of slices in to account for better result we use function to generate number of slices as per requirement for testing. Precise difference is recorded when we increase the size of slices.

We are working on text data files only each data file is of 1 kb and total dataset is around 2.0 MB. Major time difference can be calculated if data size of scale GB.

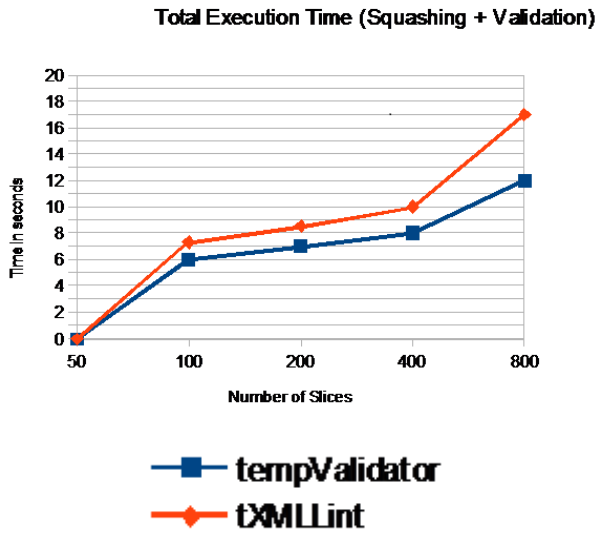


**Figure 6: Total time required for Squashing and validating by both the tools.**

We studied our result in following situations:

1) Use same number data files, squashed them and then validate. This represented in total system execution time graph.

2) Increase the number of data files (versioned slices), generated for each of 4 constraints, squashed them and then validate. And only record validating time of each constraint with variable number data slices. We varied the number of slices from 50 to 800, to examine the time scaling difference, so as the base paper does. This test perform on basic minimal system configuration environment i.e. Windows XP having processor of 2.6 GHz , having data storage capacity 64 GB, and RAM 2 GB.

## 6. CONCLUSION AND FUTURE ENHANCEMENT

In this paper, we propose the approach that will support Schema versioning for time varying XML documents which is upward compatible with XML. We have created the tempValidator tool for validating the temporal document. Also, we integrate tXSchema with a Schema aware editor, which are able to browse the base schema from which versions are created. It will also provide support for a creation of temporal document, validation for both conventional as well as temporal schema. As base paper supported the concept of squashing that integrates non temporal and temporal document together to form a representational Schema, we have implemented Unsquash policy which able to reverse the Squash output, i.e. from representational schema we can generate Schema slices back.

In the future, we will plan to improve tracking of changes in XML document, also we plan to add versioning to Xupdate, it is the language which specifies changes to an XML. By specifying how the evaluation of an Xupdate statement on an XML Schema document use to modify a bundle.

One area of future work can be optimization and efficiency concern. It would be useful to observe the impact of timestamp placement (physical annotation) and impact of parameters (logical annotation) such as evaluation window size on efficiency (document size, IO time).

## 7. ACKNOWLEDGMENTS

I take this opportunity to express my intense gratitude and deep regards to my Guide and HOD of information technology, Prof. S.E.Pawar for her exemplary guidance, monitoring and constant encouragement throughout this paper.

I would like to express deepest appreciation towards Prof. Dr. G.J.Vikhe Patil., Principal of Amrutvahini College of Engineering, Sangamner, and Prof. B.B.Borkar (ME Coordinator) who's invaluable guidance supported me in completing this paper.

## 8. REFERENCES

[1] "Adding Temporal Constraints to XML Schema" Faiz A. Currim, Sabah A. Currim, Member, IEEE, Curtis E. Dyreson, Richard T. Snodgrass, Senior Member, IEEE, Stephen W. Thomas, Member, IEEE, and Rui Zhang.

[2] C. Dyreson, H. L. Lin, and Y. Wang, "Managing Versions of Web Documents in a Transaction-time Web Server, in Proceedings of World Wide Web", New York, NY, pp. 422432, 2004.

[3] J. Chomicki, "Efficient checking of temporal integrity constraints using bounded history encoding," ACM Trans. on Database Systems, vol. 20, no. 2, pp. 149–186, 1995.

[4] J. Chomicki and D. Niwinski, "On the feasibility of checking temporal integrity constraints," Journal of Computer and System Sciences, vol. 51, no. 3, pp. 523–535, 1995.

[5] J. Chomicki and D. Toman, "Implementing temporal integrity constraints using an active dbms." IEEE Trans. on Knowledge and Data Engineering, vol. 7, no. 4, pp. 566–582, 1995.

[6] S. Y. Chien, V. J. Tsotras, and C. Zaniolo, "Efficient schemes for managing multiversion XML documents." The VLDB Journal, vol. 11, no. 4, pp. 332–353, 2002.

[7] J. F. Roddick, "Schema evolution in database systems: an annotated Bibliography." SIGMOD Rec., vol. 21, no. 4, pp. 35–40, 1992.

[8] C. A. Curino, H. J. Moon, and C. Zaniolo, "Graceful database schemaevolution: the prism workbench" in Very Large Data Base, 2008.

[9] C. S. Jensen and C. E. Dyreson (Editors), "The Consensus Glossary of Temporal Database Concepts,"February 1998 Version.

[10] Document Type Definition (DTD) language. URL http://www.w3.org/TR/REC-xml/dt-doctype,Viewed March 25, 2007.

[11] SAX project, Official website. URL http://www.saxproject.org, Viewed March 26, 2007.

[12] C. S. Jensen and R. T. Snodgrass, "Temporal Database Management," TimeCenter TR-17, 1997.

[13] R. T. Snodgrass, "The Temporal Query Language TQuel," in ACM Transactions on Database Systems12(2):247–298, June 1987.

[14] Document Object Model, W3C. URL http://www.w3.org/DOM, Viewed March 26, 2007.

[15] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, Temporal Databases:Theory, Design, and Implementation, Benjamin/Cummings Publiihing Company, 1993.

[16] https://books.google.co.in/books?isbn=3540212000.

[17] http://timecenter.cs.aau.dk/TimeCenterPublications/TR-89.pdf

[18] http://xml.silmaril.ie/parsers.html.

[19] http://www.researchgate.net/publication/222418199 Validating_quicksand_Temporal_schema_versioning_in_XSchema.

[20] http://en.wikipedia.org/wiki/Simple_API_for_XML.