



XSDyM: An XML graphical conceptual model for static and dynamic constraints



Norfaradilla Wahid^{a,b}, Eric Pardede^{a,*}

^a La Trobe University, Australia

^b Universiti Tun Hussein Onn Malaysia, Malaysia

ARTICLE INFO

Article history:

Received 29 November 2013

Received in revised form 6 June 2014

Accepted 25 June 2014

Available online 8 July 2014

Keywords:

Graphical

Modelling

XML

Dynamic constraints

Business rules

ABSTRACT

Data modelling is not only important to visualise the structural schema of data, but also to show the intended integrity constraints. In this paper, we propose a modelling approach called XML Static Dynamic Modelling (XSDyM). While a text-based schema definition is often the most common method used to describe XML, graphical modelling is more accepted as it is capable of visualising the schema definition more effectively for the reader. Conveying the dynamic constraints on XML graphical model requires a special treatment as the constraints basically comprehend the state transitions. It is important for an XML modelling to keep the basis as precise as possible to satisfy the nature of XML and at the same time be able to represent the constraints in an effective way. Using the XML tree-based modelling as the basis of the work, we proposed our own approach to convey the state transitions of the constraints, where it is inspired from the well-known state diagram and adopt some useful features of ORM modelling. We evaluate the correctness of our proposed modelling using a model which involves the checking of model transformations between the modelling and the equivalent XML schema languages.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

XML allows us to model information systems in a natural and intuitive way and to express information to match the way we do our business. In the field of XML databases, modelling the structure of the data and their integrity constraints is an issue that should be considered in XML research. This is because XML has a unique structural nature that differs from any other type of database format which usually can be modelled using popular modelling tools like the Unified Modelling Language (UML), Entity–Relationship Diagram (ERD), and Systems Modelling Language (SysML). Another modelling tool that has become popular recently is Object Role Modelling (ORM). Every modelling approach has been proven to have its own strengths to handle different rules and constraints.

Data modelling is important not only to visualise the structural aspect of the data, but also to show the intended integrity constraints or semantics of the data. A large body of work has been conducted e.g. [4,33,18] which particularly discusses not only the structure but also the integrity constraints that XML data should have with regard to business rules. XML data is often viewed as edge labelled graphs or trees as the natural essence of the semi-structural model. But, as far as we are concerned, none of the work on expressing XML constraints proposes a proper modelling approach to convey the constraints.

A lot of effort has been expanded on expressing and maintaining integrity constraints in XML. According to Ref.[23], constraints can be classified as static and dynamic. Almost all the existing works on XML concern static constraints and discuss the condition of constraints at any time space. A dynamic constraint is another type of constraint to express the condition that involves facts/requirements between two and more states during their transition within a given state space [30]. In a managerial context, a dynamic constraint can be seen as representations of ‘real world’ constraints and business rules [7]. We suggest that, the dynamic constraints come from soft rules while static constraints come from hard rules (both from business rules). In terms of business rule modalities [9], dynamic constraints are mostly treated as deontic¹ rather than alethic² constraints.

Modelling dynamic constraints, particularly in XML, is a challenge that we need to face. Different to static constraints, we need to focus on visualising the state transition properties instead of only static requirements at any particular state of time as is required in static constraints. At the same time, the modelling should be able to convey both types of constraints as dynamic constraints might be degenerated from static constraints. Research in ORM is one of the most active areas of research in modelling different data structures and different types of constraints, including dynamic constraints. Unfortunately, the

¹ Deontic rules impose obligations, which may be violated, even though they should not. For example: it is obligatory that each employee is married to at most one person or no smoking is permitted in any office.

² Alethic rules impose necessities, which cannot, even in principle, be violated by the business, typically because of some physical or logical law. For example: each employee was born on at most one date.

* Corresponding author at: Department of Computer Science & Computer Engineering, La Trobe University, Bundoora VIC 3083, Australia.

E-mail addresses: nwahid@students.latrobe.edu.au (N. Wahid), E.Pardede@latrobe.edu.au (E. Pardede).

effectiveness of ORM can be overshadowed by the unnecessary complexity of the model if it is to be implemented in XML. ORM is constructed from the connections of entities and roles. Therefore, in order to convey XML, we have to have roles for each connection between nodes. Ref. [5] showed that although ORM can specify a wide variety of data constraints, including mandatory roles, uniqueness, subsets, exclusion, frequency and ring constraints, it fails to describe integrity constraints, which play an important role in maintaining XML data. It is also hard to capture some specific XML characteristics, such as order and hierarchical structure. Therefore, based on this intuition, we believe that ORM is not the best tool to visualise XML and its constraints.

Work has been conducted on mapping the ORM design into a compatible XML Schema [3] in order to conform to the XML specifications and nature. It is possible to obtain a useful matching XML based on the ORM, but it requires extra effort (based on the proposed work) including ‘anchoring’ and ‘fact type grouping’ between the elements and at the very least, requires the creation of a dummy element to become the ‘root’ of XML tree. They also addressed a few limitations or incompatibilities of the ORM and XML Schema. Therefore, based on this discussion, we make an assumption that, to ensure compatibility between the two of them, steps must be followed since both are different in terms of structural strength and properties.

It is well known that XML is typically visualised using a tree-based or a graph-based modelling approach with the nodes conveying elements and attributes while some vertices show connections between the nodes as in Fig. 1. Without doubt, this is the most common method used to visualise XML. We would like to maintain this simplicity and at the same time be able to express more complex properties into the model such as conveying the internal transition of the node states.

There is quite a large body of work on XML conceptual modelling. According to Ref. [5], the work can be categorized into three; i.e. (i) based on ORM/NIAM [3], (ii) based on ER, e.g. ERX [24], XER [28], ERex [19] and XSEM [21], (iii) based on a tree structure or graph, e.g. ORA-SS [16] and GOOSSDM [26] and (iv) based on a Semantic net [25]. Even though there is a selection of modelling for XML and a few are based on tree/graph modelling, unfortunately none of the approaches provides a way to convey a good range of integrity constraints, particularly for dynamic constraints, although a few have the ability to model static constraints, like inclusion dependencies and functional dependencies.

Research on collaborating the ‘state diagrams’ [29] and modality is not new. It is actually well-known that the family of modal logics is defined via graphs called Kripke structure.³This is basically a graph whose nodes represent the reachable states of the system and whose edges represent state transitions. A labelling function maps each node to a set of properties that hold in the corresponding state. Temporal logics are also traditionally interpreted using Kripke structures [13].

In this paper, we propose a modelling approach which will take the XML tree-based modelling as the basis approach. Using this basis, we retain a few useful features of ORM into our work as it has rich modelling notations specifically for modelling the dynamic constraints. We proposed the transition features on the modelling inspired from the state diagram to visualise the dynamic constraints of the data. We called it XML Static Dynamic Modelling (XSDyM). Therefore, the main contribution of this paper is the proposal of a special modelling approach which is useful for static and dynamic constraints in XML data.

1.1. Outline

The outline of this paper is as follows: Section 2 gives the background of ORM and state diagram. In Section 3, we defined our approach together with the related definitions and details. In Section 4, we present the transformation rules between the modelling and XML grammar languages. In Section 5, we show the correctness checking of our proposed approach

³ A variation of non-deterministic automaton proposed by Saul Kripke[17] used in model checking to represent the behaviour of a system.

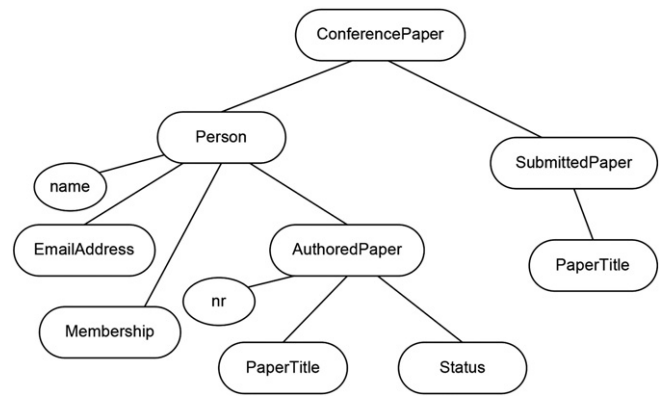


Fig. 1. Basic XML data on tree modelling.

based on the work in Ref.[20]. In Section 6, we provide a discussion of the strengths and weaknesses of both ORM and our modelling approaches. We conclude our paper in Section 7.

2. Background

Two concepts are essential in order to understand this paper i.e. the notations of ORM and related XML modelling, and the definition of state diagrams used in the paper. In later sections, we detail the modelling of dynamic constraints using both approaches. We briefly introduce them in this section.

2.1. Object Role Modelling (ORM)

We assume the readers are familiar with ORM so here we briefly mention some basic features of the modelling environment. Interested readers are invited to read [8,6,10] for the full specification of ORM and ORM 2.

Object Role Modelling (ORM) is a conceptual modelling method that allows the semantics of a universe of discourse to be modelled at a highly conceptual level and in a graphical manner. It has been used commercially for more than 30 years as a database modelling methodology, and has recently become popular in many areas such as ontology engineering, the modelling of business rules, XML-Schemas, data warehouses, requirements engineering, and web forms. ORM has an expressive and stable graphical notation. It supports n-ary relations and reification, as well as providing a fairly comprehensive treatment of many “practical” and “standard” business rules and constraint types. Furthermore, compared with, for example, EER or UML, ORM’s graphical notation is said to be more stable since it is attribute-free; in other words, object types and value types are both treated as concepts. This makes ORM immune to changes that cause attributes to be remodelled as object types or relationships. ORM also offers the automatic verbalization of the diagram into pseudo-natural language sentences. From a methodological viewpoint, this verbalization capability simplifies communication with non-IT domain experts and allows them to better understand, validate, or build ORM diagrams.

In Fig. 2.1, we show a few basic components of ORM and then we show a minimal example of an ORM model in Fig. 2.2. Based on the example, we have some relationships between entities Country, Person, University and Gender shown in soft rectangles. The relationship between the entities is recognized by the existence of role boxes accompanied by the role names to connect the entities. For example in the figure, a connection between the entities Person and University is a role “graduated from”. As mentioned earlier, ORM offers a wide range of notations to express business rules and constraints. Some are uniqueness constraints, mandatory role constraints, set-comparison, exclusive-or, frequency and value constraints, ring constraints, etc. To illustrate the value constraints, we show in the figure the textual (value) of ‘M’ and ‘F’ noted in curly braces which means that a Gender can only exist between these two values. On the other hand, the dark

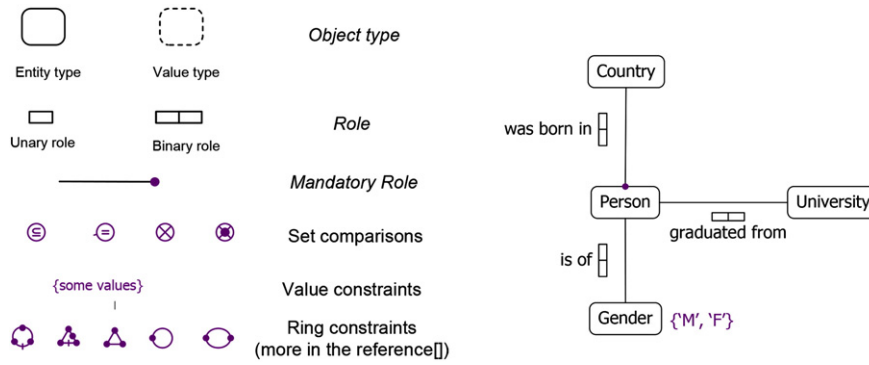


Fig. 2. (2.1) ORM basic notations. (2.2) ORM example of an employment information model.

circle notations attached on the entity Person illustrates the mandatory role in its connection with Country.

2.1.1. ORM and dynamic constraints

Based on the proposed work by Halpin in Ref. [12], we require the use of textual constraints as the language to convey the dynamic constraints. Textual constraints may be noted on the diagram by footnote numbers, with the textual reading of the constraints provided in footnotes that can be both printed and accessed interactively by clicking the footnote number. There is still no standard to define dynamic constraints and we have to rely on the textual constraints. We show an example of using textual constraint in Fig. 3.

As XML is simply formed by connecting element nodes and attribute node, adding additional textual information on the model may appear awkward. Therefore in [31], we shows our approach of conveying dynamic constraints using ORM, by simply showing the transitions of states using a unary role called updates to show that there is a possible transition within that particular entity (refer Fig. 4). Since the work is on object composition, we differentiate the object entity using an old notation of ORM. As the constraints become bigger, the modelling will see relatively complex roles and relationships in order to model dynamic constraints.

Hence, in our opinion, to model an XML document using ORM is a daunting task as it is necessary to create roles on every connection of nodes, hence the model will become unnecessarily complex more than the normal XML tree model.

2.2. The state diagram

A state diagram is a type of directed graph used in computer science and related fields. They are used to give an abstract description of the behaviour of a system. This behaviour is analysed and represented in a series of events, which could occur in one or more possible states. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

State diagrams have been popular to graphically represent finite state machines. In more specific work, the idea of state diagrams has also been successfully used to illustrate the states an object can attain as well as the transitions between those states in the Unified Modelling Language (UML), where they are also called a state machine diagram or state-chart diagram.

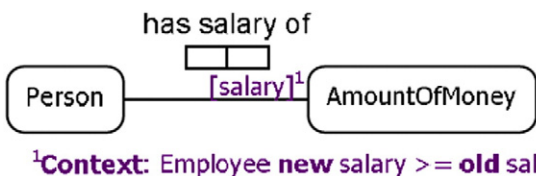


Fig. 3. An example of dynamic constraint modelling using ORM by Halpin [12].

Formally, we can define a state diagram to consist of a set of states, S , a set of actions A and a transition function δ that maps out the edges of the graph, i.e. shows how actions result in state changes. Each edge of the state diagram shows what happens when you are in a particular state s and executes some actions a , i.e. which new state s could be in.

An input δ is a pair (s, a) of a state and an action. In some applications, there is only exactly one new state that could result from applying action a in a state s . For these applications, we can give δ the type signature $\delta : S \times A \rightarrow S$. However, in some state diagrams, it might not be possible to execute certain actions on creation states. When we need to support these possibilities, each output of delta must be a set of states. So the type signature $\delta : S \times A \rightarrow P(S)$.

State diagrams of various sorts, and constructs similar to state diagrams, are used in a wide range of applications. Therefore, there are many different sets of terminologies for equivalent and/or subtly different variations on this idea. In particular, when a state diagram is viewed as an active device, i.e. as a type of machine or computer, it is often called a state transition or an automaton. In automaton, start

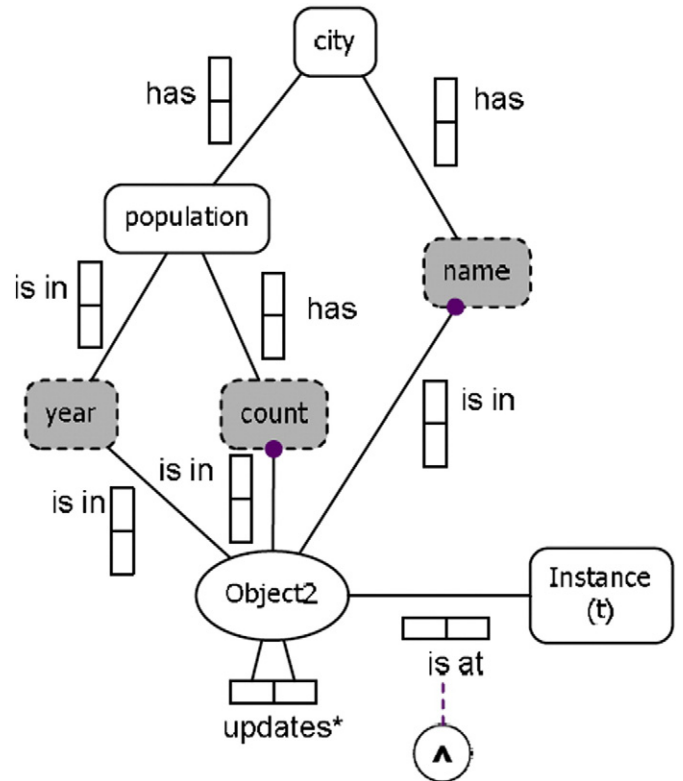


Fig. 4. An example of our refined ORM to show a dynamic constraint of XML in [31].

states are also known as initial states while end states are usually called end states or goal states.

Fig. 5 is a state diagram of automaton M. It has three states, labelled q_1 , q_2 and q_3 . The arrow pointing at it from nowhere indicates the start state. The accept state q_2 is the one with a double circle. The arrows going from one state to another are called transitions.

3. Proposed modelling approach

Our proposed approach is inspired by the ability of state diagram modelling to show state transitions in an easy way, but, the aim of our paper is not to follow the semantic flow of automata as has been discussed in a previous work on automata, logic and XML [22,27]. We are aware of the fact that we need to follow some unique procedures to achieve an interchangeable model of an XML tree and the automata model. Therefore we would like to make a restriction that our proposed modelling approach only takes the essence of the state diagram approach and is based on the intuition of the tree modelling approach with the existence of some events (transition).

The intention of our proposed XML modelling approach is to develop a method where the user does not have to struggle between the transformations of the normal XML-tree model to other modelling. The goal is to keep the modelling based on the normal XML-tree model but at the same time, to offer more components to allow the expression of constraints or rules on it, especially for dynamic constraints. We will use the following definitions to support our notions of model.

Definition 1. Let tree $T = (N, E, r, \Sigma, \lambda)$ be the XML tree in document D .

- N is a set of nodes.
- $E \subseteq N \times N$ is a set of edges.
- r is the root node.
- Σ is the set of element name appearing in D .
- There is a subset of Σ that associates with attributes *att*.
- λ is the labelling function that associates an element name with each node other than the root, where $\lambda: L - \{r\} \rightarrow \Sigma$.

Therefore, we generalise the definitions above to suit our approach and define the XSDyM tree as follows:

Definition 2. XSDyM tree, T_M is constructed from 6 tuples where $T_M = (N, E, r, \Sigma, \lambda, (S, \mathfrak{R}, \delta))$.

- $(S, \mathfrak{R}, \delta)$ is a tuple set, which comes together in the case of the transition of states.
- S is the states. An $n \in N$ associates with state S where $n_s \in S$ is the state of n in the time space.
- \mathfrak{R} is the set of rules and δ is the transition function, $\delta = S \times \mathfrak{R}$, i.e. $\delta: S \times \mathfrak{R} \rightarrow S'$.
- For each transition function δ there must be at least one rule $\in \mathfrak{R}$.

If there exist no dynamic constraint(s) in tree T_M , then $(S, \mathfrak{R}, \delta)$ can be an empty tuple where $(S, \mathfrak{R}, \delta) = \epsilon$.

Definition 3. Let TS be a time space, where $TS = \{t_i | 0 < i < x\}$. x is a finite number and t_i is a discrete time point. The transition function δ as in Definition 3 involves the move from the current state at a current time

point t_i to a new state where S_{t_i} to $\{S_{t_i} | S_{t_j}\}$ where t_j is the new time point and $j = \{(i + x) | (i - x)\}$. Therefore, $S' = \{S_{t_i} | S_{t_j}\}$.

In most common conditions, dynamic transition might occur in local states, i.e. within the same node, n that is, the transition function is denoted for state $n_{S_{t_i}} \rightarrow n_{S_{t_j}}$. An example can be found in the earlier discussion. There might be cases where we want to constrain the transition between different nodes, from a state to another state i.e. $n_{S_{t_i}} \rightarrow n'_{S_{t_j}}$. For instance the dynamic Functional Dependencies involve dynamic dependencies of two different sets of nodes on the left and the right of the equation (see [32]).

As explained in the Background section, very limited research has been conducted on dynamic constraints, and currently ORM is the most promising approach for dynamic rules. Therefore, as defined earlier, our approach is a unification of the XML tree model with the essence of the state diagram and utilizes some ORM components to strengthen our model.

3.1. The modelling

3.1.1. An example conference paper data model

The implementation discussed in this paper is based on the “Conference Paper” example presented in the Introduction of this paper, i.e. see Fig. 1. The model of *Conference Paper* consists of two main subtrees, i.e. *Person* and *SubmittedPapers* subtrees. For each *Person* element, it contains the *Authored Paper* and *Email Address*. *Authored Paper* also includes *Paper Title* and *Status* of the paper as the children. Each *Person* will also has a *Name* attribute. On the other hand, *SubmittedPaper* offers to list down all the *Paper Titles* that have been submitted.





Using this example, we present the data and constraint modelling using our notations as summarized in Tables 1 and 2. It is important to mention that we also extend this data model with additional relevant entities to suit our discussions below in particular for dynamic constraint modelling.

3.1.2. Entity type and relationships

As mentioned in Ref. [1], semi-structured database instances are generally seen as a labelled graph. Labels are sometimes attached to edges and sometimes to vertices. The labels are meant to convey information about logical data organization. Therefore, it is obvious that we opt for the second choice w.r.t. labels corresponding to XML tags.

Table 1 presents the notations to convey the most basic component of the XML tree, i.e. conveying the nodes. In our approach, we treat the entities, i.e., the element and attribute nodes using different components. The soft rectangle is used for elements and the oval is for attributes. Therefore, in Fig. 1, it is shown that the element nodes are *Conference Paper*, *Person*, *Authored Paper*, *Email Address*, *Paper Title*, *SubmittedPaper* and *Status*, whereas there is an attribute for the element *Person* which is *Name*.

Table 1
Basic XSDyM modelling components.

Component	Description
<i>Entity type</i>	
	Element node of XML
	Attribute node of XML
<i>Relationship</i>	
	Common element to element/attribute connections
	Node referencing, or dependencies

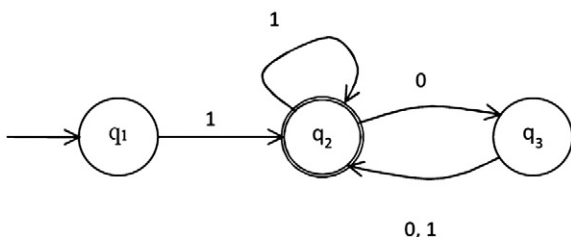

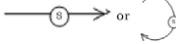







Fig. 5. Well-known state diagram for automata theory.

Table 2
XSDyM modelling component for dynamic constraints.

Component	Description
Relationship arrow	
	The 's' in the circle marks the state transition i.e. to mark the transition function δ
	Arrow connection line to another
C:n	To be used together with state's circle, to marked the restriction on the state counting or the time-frame.
Dynamic direction	
	To mark future direction of dynamic state
	To mark history direction of dynamic state
	To mark both future and history direction of dynamic state
Association	
	Union symbol to mark the union of nodes or a node is a member of an object
	Connecting the association-ship symbol
ALL	All of child/parent or sibling
FIRST	The first of child/parent or sibling
LAST	The last of child/parent or sibling
x	x number of child/parent or sibling
CHILD/PARENT	To mark child or parent of the aim node
SIBLING	To mark sibling of the aim node
State constraint expression	
i. State position, S	
S	Let sequence of states s_{i-1}, s_i, s_{i+1} Current state, s_i
\hat{S}	Goal state, depends on the dynamic direction i.e. s_{i-1} or s_{i+1}
ii. Changeability	
DEL	Delete operation
INS	Insert operation
REP	Replace operation
iii. Special restriction (to support changeability constraint)	
ONLY	To mark uniqueness constraint
&&	AND operator
	OR operator
!	NO operator

Meanwhile, the second section of the table lists all the connection arrows in the modelling. The standard connection line to connect between entities and attribute in the model will be straight lines, while an arrow with a dashed line is used to show the referencing or dependencies between nodes where necessary. The connection arrows shown in Table 2 will be used particularly for dynamic constraints and will be discussed in the next subsection.

Fig. 6 shows a minimal example of static constraints. Since this paper only focuses on dynamic constraints, we will avoid further discussion on the modelling of static constraints. It is shown in the model that, the *Paper Title* element node under *Person* sub-tree has made a value reference to *Paper Title* under *SubmittedPaper* using the dashed line arrow, i.e. it makes some value dependencies to another node value.

3.2. Expressing the dynamic constraint

To provide a way to express the constraints as precisely and simply as possible, we propose a standard grammar to verbalize the constraints. This will allow us to mark the constraints explicitly on the model whenever necessary and is more feasible than fully depending on textual constraint as in ORM. Based on Definition 3, the rules, \mathfrak{R} will mark the

constraints as the transition function δ occurs. The following is the template for expressing constraints on transitions.

- **Rule Context Transition, RCT** for rules \mathfrak{R} is the form to specify the details of transition context and the related state changing requirement. It can be written in two ways, i.e. (i) using *Full Term*, “context name $s\hat{O}s$ ”, or (ii) using *Simplified Term*, “ $s\tilde{O}s$ ”. *context name* refers to the element/attribute name and \tilde{O} is the connecting operator. The default for \tilde{O} operator is “to”, but it might come from any relevant binary operator. The *context transition* \mathfrak{R} construction is inspired from the ‘constraint expression’ in ORM. Refer to Fig. 7 for an example of the modelling. Both figures in Fig. 7 can be described as a model to convey the dynamic constraint on *Salary* value, where each updated value must always be a higher value than the current *Salary* value, i.e. *Salary* value at the next state must be higher than at the current state.

- **Grammar of the rule of \mathfrak{R} .** If the complete rule is taken as a string which can be derived from \mathfrak{R} of CFG⁴ below:

```
<rule> → RCT
<rule> → RCT '|' <ext>5
<ext> → <ext> ; <ext> | enumeration sequence | changeability | dependability
<ext> → ∈.
```

- ◇ *enumeration sequence* is to show the sequence of the allowed update input based on the existing state. Fig. 8 is showing the example of enumeration sequence of valid *Status* over time. We listed all the valid conditions in the rule expression. For instance in this case, valid *Status* changes include from Single to Married and from Married to Divorced, whereas state transition from Single to Divorced is not an allowed transition.

- ◇ *changeability* specifies update operation constraint, for example, we need to show that some elements cannot be deleted or cannot be updated on the query update operation. Refer to Fig. 10 where it shows a case where deletion is not allowed on the nodes marked by !DEL.

- ◇ *dependability* conveys any element node additional information about referencing or dependencies. But, if the constraint is minimal, the user can simply show the referencing using the dashed line arrow (please see Table 1). For example one wants to denote a constraint is specifically Functional Dependencies, it can be marked by the “FD” and can use the “key” or “non-key” for referencing. Details on this will be captured in the future work.

- **Rule expression of \mathfrak{R}** can be made using the template, [*<rule>*]. This is the point where the user can choose to write the constraint as a reference by using a footnote writing style as used in ORM modelling. But, to allow this, we need to use the *Full Term* of context transition, RCT.

Using the template is rather straight forward. For example, to convey a dynamic constraint “*updating an age to smaller value is not acceptable*”, can simply be written as [*age : s< \hat{S}*]. In this case, *age* is the *context name* while *s* and \hat{S} are the standard to note the current state and the next state, and < is the binary operator.

We include in this paper examples of using the modelling approach for different requirements.

3.3. Dynamic constraint special features

3.3.1. Dynamic direction

In Ref. [23] the authors claimed that dynamic transition can occur in different directions of temporality, i.e. future-based or history-based. The first is to show the constraints of the next state while the second

⁴ Context Free Grammar.

⁵ RCT '|' <ext> produces RCT | <ext>, i.e. '|' stands for a vertical bar, not for an OR logical disjunction.

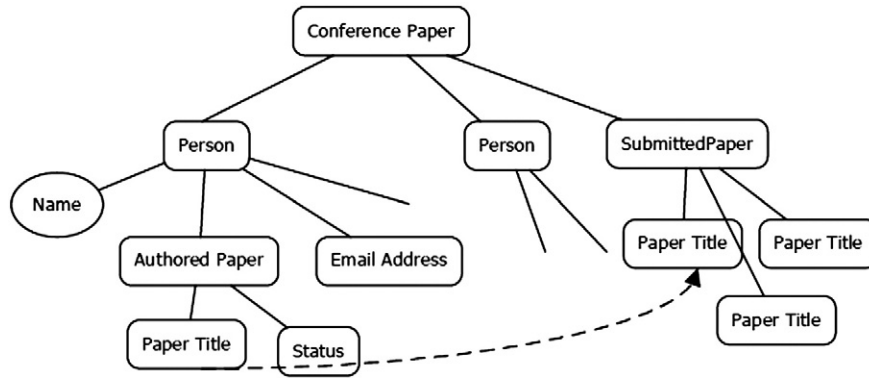


Fig. 6. Static referential constraints in XML.

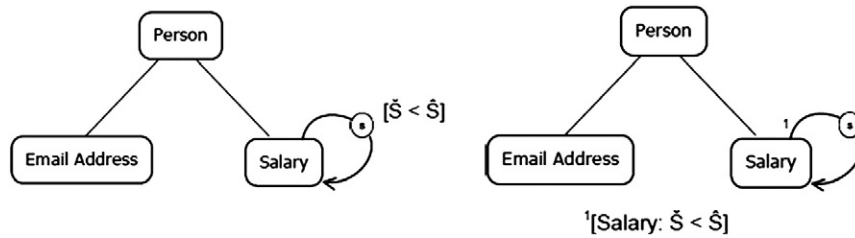


Fig. 7. Different ways of conveying local dynamic constraints.

is to show the constraints based on the previous state. Therefore, we differentiate these two conditions by using different directions of the state arrow. Hence, in $s\hat{O}\hat{s}$ notations, we have to make sure that s is always the state of where the state transition started, while \hat{s} is the state where it moved to regardless of whether it is a future state or a history state. Notice that to convey the transition of states, we use a special arrow marked with an 's'.

As seen in Fig. 9, we use an anti-clockwise arrow to express a dynamic constraint on *Salary* which is based on the history state. The constraint in the model is saying that the current *Salary* must be always the same as with the previous ones for each transition which occurs. On the other hand, Fig. 7 shows a future-based constraint where the figures use clockwise arrows on the models.

The dynamic directions accommodate Definition 3, where the new state can be at t_j where $j = \{(i + x)|(i - x)\}$. Therefore, if $j = i + x$ this means that the dynamic constraint is future-based, whereas if $j = i - x$ it is said to be history-based constraints.

3.3.2. The time frame

In the case of temporality, one might want to bind the constraints to a certain count or time-frame. In our modelling, we use the $C:n$ to restrict the state count and $C:t$ to restrict the time-frame of the transitions. For example, $[time : S < \hat{S}]C : n = 3$ says that the counting of the constraints is restricted to three states only, whereas, $[time : S < \hat{S}]C : t = \{0800-1000\}$ specifies that the temporality of the constraints is between periods 0800 and 1000. We do not set any restriction on the

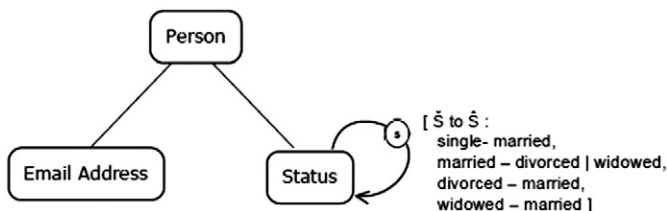


Fig. 8. Example of enumeration sequence on state transition.

time unit/measurement as it may differ according to business requirements.

3.3.3. Constraint locality

In an XML environment, transitions of dynamic constraints may occur locally within a specific node, i.e. *local constraints*, or could also occur between different nodes, which we refer as non-local transitions, i.e. *non-local constraints*. We believe that our modelling approach is capable of conveying both types of constraints. The following discussions give a few related examples of using our modelling components, as summarized in Table 2.

3.3.3.1. *Local constraints*. A local dynamic constraint is the condition where the constraint is meant for one specific node or a group of nodes without a “semantic dependencies” between them. Figs. 10 to 15 show a variety of local dynamic constraint models.

Two examples of local constraints are the single node constraints [30] and the cumulative node constraints [31]. All the figures in Fig. 7 to 9 are some examples of single node constraints. On the other hand, Fig. 10 shows a modelling example of cumulative node dynamic constraints as proposed in Ref. [31].

As the reader can see, the involved nodes are connected together using dashed lines, which model the cumulativity of the nodes (treated as an object in Ref. [31]). An element node, a , is connected to the union symbol with a ‘mandatory entity’ component (dark black dot) where it

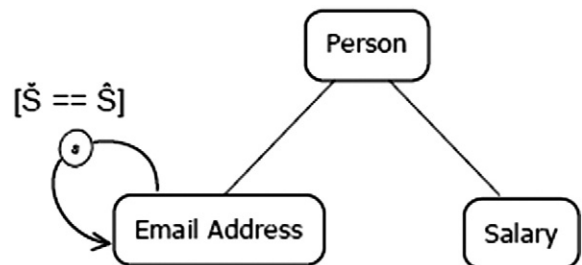


Fig. 9. History-based dynamic constraints.

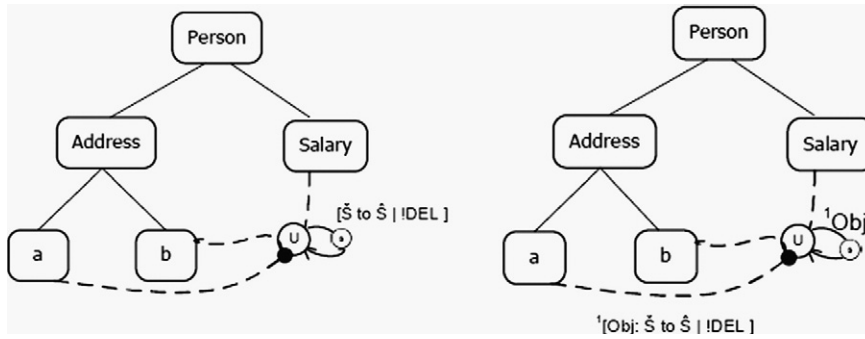


Fig. 10. Cumulative node dynamic constraints.

has different strengths on the required constraint to the data being updated. Notice that this other feature is inherited from the ORM (mandatory roles). The diagram on the right has similar semantics to the one on the left, except that the constraint is written in footnote-style, therefore a reference name needs to be created to represent the node collection. In this case we use *Obj* as the reference in the model, hence the same name is used in the referred constraint argument.

Fig. 11 shows an example of cumulative node constraints on a subtree. The words ALL CHILD in this example marks such a condition, where the constraint applies on *Address* and all the children of the node.

Notice that in Fig. 3, we show our refined ORM modelling to suit the dynamic constraints in [31]. In conjunction with this, we remodelled the same XML constraint using our approach in Fig. 12. As can be seen, in addition to the removal of role usage in the modelling, a few unnecessary element nodes have been removed from the model. The node up-

date and instance has been simplified using our ‘dynamic transition’ notation. Therefore, our modelling is a much simpler and more precise approach compared to modelling in ORM.

All the specifications above are used on the local constraints where all the dynamic constraints exhibit no dependencies on nodes other than itself or its own object, i.e. everything is meant for its own node and the states.

3.3.3.2. *Non-local constraints.* We propose a non-local constraint as another case of dynamic constraint where it is constructed from the nodes within the same or different trees and there is at least one semantic reference or dependency between the nodes. An example of a non-local constraint is shown in Fig. 6 with the assumption that the reference is at a different state to create a dynamic environment.

Given a set of node *X* and a set of node *Y*. *X* and *Y* can form a non-local constraint if the δ function involves dependencies, implication, referential, or special binary operation, i.e. monotone “semantic relation” between the two sets. In 3, the tuple $(S, \mathfrak{R}, \delta)$ is a tuple set where it comes to convey the generic type of dynamic constraints which is based on state transition (or called transition constraints in Ref.[11]), i.e. in this case δ is a transition function. In the non-local constraint the problem does not particularly involve the transition of states where $\delta = S \times \mathfrak{R}$, instead, δ carries the integrity of set *X* to set *Y* but at different states. For example if there exist functional dependencies between *X* and *Y* and we would like the constraint to be extended to another time state, it becomes a dynamic constraint of *X* (at the current state) connected to *Y* (at the new state) by functional dependencies, instead of a transition.

Another example of modelling non-local constraints is shown in Fig. 13. Let’s consider, in this case, that there is an *inclusion dynamic constraint*⁶ of collective nodes *X* and *Y*. It is shown in the model that *X* is constructed from nodes *a* and *b* while *Y* is constructed from *a*, *b* and *Salary*. The rule constraint can be expressed using the notations $[X, Y : S_X \subseteq S_Y]$ in a footnote style. In this case, $S_X \subseteq S_Y$ should be read as *at current state S of X, X has an inclusion dependencies on the next state of Y*.

The static behaviour of the dependencies could also be conveyed by using the notation of $S_X \subseteq S_Y$. This is obviously shown by the occurrence of *S* in both state notations.

As mentioned in Ref. [5], most existing conceptual modelling particularly for XML has the main limitation of being short of mechanisms to specify more data semantics, including integrity constraints. We admit that in some circumstances, the graphical model is not the best option for modelling the constraints because of complexity. However our approach provides an alternative to visualise the constraint on a graphical model.

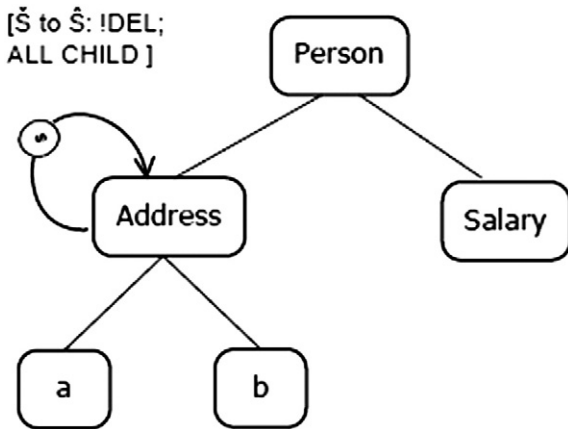


Fig. 11. Cumulative node constraints in a subtree.

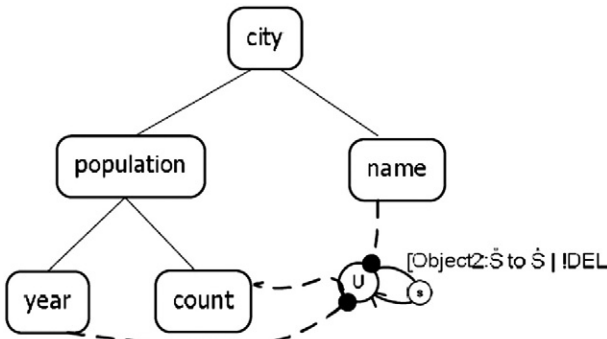


Fig. 12. Modelling Fig. 4 into our modelling approach.

⁶ $X \subseteq Y$.

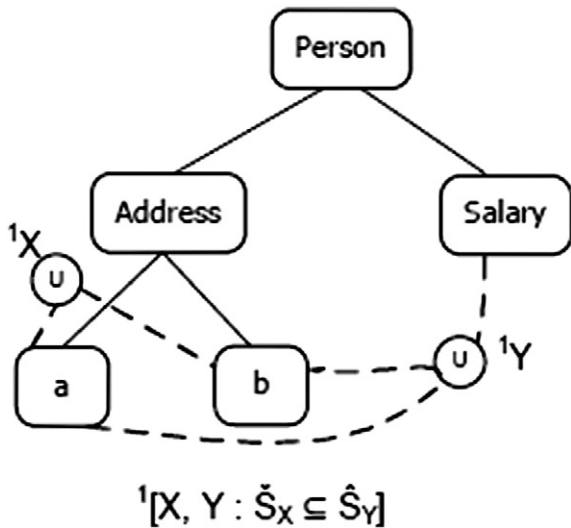


Fig. 13. Inclusion dependency example in a dynamic context.

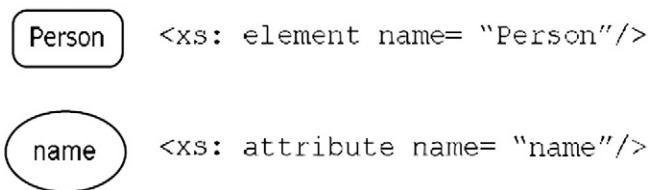


Fig. 14. Representation of tree nodes.

In a previous research of dynamic constraints in relational database or object-oriented database, states S are usually referred as the state of attributes' value on an entity. It can be seen in the XML database as the node's value (or text), i.e. at the leaf node instead of the node itself as a whole (which may be a mixed-content type). For instance, in the earlier example of *Salary* transition, the discussion is indirectly about the state transition of the *Salary* text value, i.e., the value held by *Salary*. Another example is on a person's *Status* where the transition constraint actually refers to the value held by *Status*, e.g. single and married. Other similar work such as on temporal database constraints also regards that the temporality of data is on the versions of data values within a specific time-frame. Therefore, we would like to address that this is the most common condition of the dynamic constraints that could exist in the XML database too.

Nevertheless, in other cases, for example in Fig. 11, the constraint can be read as 'no deletion' or 'no value deletion' allowed on the nodes, i.e. all the nodes rooted at *Address* cannot be deleted. Therefore in an XML environment, determining the dynamic constraint is not restricted only on the value of the element node.

4. Transformation between XSDyM and XML grammar languages

Given an XML Schema Definition (XSD) or Document Type Definition (DTD), an equivalent XSDyM can be constructed, and vice versa.

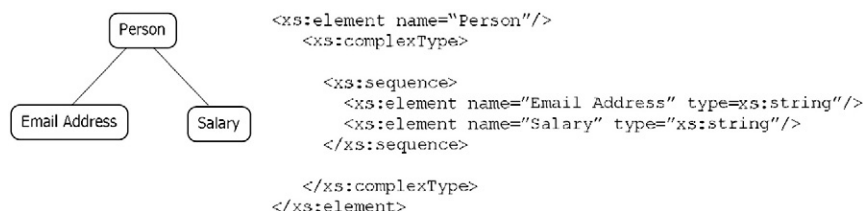


Fig. 15. An element node with children in modelling.

In this section, we will describe generic transformation rules to translate an XSDyM model to an XML grammar using XSD. Compared to the translation of the model to XSD alone, we have to consider the additional features of XSDyM, i.e. the dynamic constraints. As defined above, we denote the XSDyM tree as $T_M = (N, E, a, r, \Sigma, \lambda, (S, \mathfrak{A}, \delta))$. The main intuition behind our translation algorithm is the assumption that our XSDyM is a complete tuple, i.e. $(S, \mathfrak{A}, \delta) \neq \epsilon$. Therefore, we make another assumption that the dynamic constraints will be expressed on top of the existing schema i.e. using a supporting language for instance, Schematron.

Our algorithm can be explained in two stages: (i) *initialization*, and (ii) *constraints handling*. In initialization we look into the types of entities involved in the model, i.e. the element nodes, or attribute nodes. We also look into the relationship types and translate them. This is the basis of an XML tree. In the second stage, we handle the dynamic constraints respectively.

4.1. Initialization

In the initialization stage, we mainly work on the basis of the XML tree structure, i.e. the entities, some basic relationships and the hierarchical structure. As mentioned, at this stage, the expected transformation language is XSD as it defines the structure of the XML. Therefore, we do the following:

- Rule 1 The root of tree will become the root of XSD.
- Rule 2 Transform entity node into *element* or *attribute* in XSD.
- Rule 3 Transform entity names into a valid naming convention following XML elements and attributes.
- Rule 4 If an element node has at least one child node, the node is *complexType* in XSD.
- Rule 5 A leaf node must be a *simple type* in XSD.
- Rule 6 A static reference in XSDyM will be expressed as a *complexType* in XSD. For example, a reference of Paper Title to Paper Title as shown in Fig. 6 can be defined using key/keyref. The equivalent representation in XSD can be given in Fig. 16.

Using Rule 1 to Rule 6, we construct the basic schema of XSD. Further details *w.r.t.* XSDyM and XSD specifications are beyond the scope of our discussion in this paper. The next section will discuss the transformation rule of dynamic constraints from the model.

4.2. Dynamic constraints handling

One of our intuitions for transformation rules is that the dynamic constraints will be conveyed on top of XSD instead of in XSD itself. It is motivated by the fact that dynamic constraints typically do not come from concrete business ruleconstraints, which are expressed at the data layer of an information system. In this paper, we use Schematron as a medium to convey the constraints, therefore the following transformation rules are *w.r.t.* the transformation of dynamic constraints on XSDyM to Schematron schema, conforming to the tuple set $(S, \mathfrak{A}, \delta)$ as in Definition 3. The selection of schema language follows our proposed work in Refs. [30] and [31]. But, our approach is not limited to these schemas.


```

<xs:key name="title" >
  <xs:selector xpath=".//Authored Paper" />
  <xs:field xpath="Paper Title" />
</xs:key>

<xs:keyref name="titleRef" refer="title">
  <xs:selector xpath=".//Submitted Paper" />
  <xs:field xpath="Paper Title" />
</xs:keyref>

```

Fig. 16. Node referencing in the modelling.

```
<sch:pattern name="Local dynamic constraint">
```

Fig. 17. Constraint reference name.

Determine *locality* of the constraints. If it is a *local constraint* follow Rule 1 to Rule 9. If it is a *non-local constraint*, proceed to Rule 10. Iterate the rules accordingly.

- Rule 1 Set pattern name in Schematron as local constraint “reference name”.
- Rule 2 Set `rule context="target"` as the default of rule context in Schematron if it is a single node constraint.
- Rule 3 Node name (in the case of local constraint) will be used for checking in the test attribute of `rule` element in Schematron.
- Rule 4 The `and` in *context transitions* can be written in Schematron as `oldstate` and `newstate` in test attribute of `asset` element (Refer to Fig. 20). (See Figs. 17– 19.)
- Rule 5 The *context transition* is tested in the `assert test` in the Schematron (refer to Fig. 21).
- Rule 6 The enumeration sequence is tested individually in different branches of the `<assert test...>` but under the same rule `context` in Schematron and a related tested component can be joined using operator AND.
- Rule 7 Each element of the component in [`<rule>`] is individually tested in different branches of the `<assert test...>` but under the same rule `context` in Schematron.
- Rule 8 The intended meaning of transition constraint can be written as text message in the `assert test` element. For example, in Fig. 21, the error messages are “Salary can never decrease” and “Not possible”.
- Rule 9 The union of the nodes in the model can be translated using object creation in Schematron. In this case, we describe the `name` attribute of `pattern` element as the object name, while `name` attribute of `context` as the involved node name. Additional attribute names are added to express extra information on the node. This style of schema is based on our proposed cumulative node constraints in [31], where we also deploy a file called `delta` file in the checking. In this environment, we create a few additional attributes on Schematron to support the constraint expression. The `assert test` can be ignored accordingly if the constraint is only to control the existence of nodes.
- Rule 10 Dynamic dependency constraints can be achieved using Rule 6 of the initialization stage and the manipulation of object creation of Rule 9 in this section. For example, if there exists, a functional dependency of $(X) \rightarrow (Y, Z)$ the involved keys will be nodes X, Y, and Z, while nodes Y and Z are constructed as objects in Schematron.

```
<sch:rule context="target">
```

Fig. 18. Default rule context.

```
sch:assert test= "name = 'MaritalStatus'
```

Fig. 19. Node name in Schematron.

```
oldstate= 'married' AND newstate=('divorce'|'widow')
```

Fig. 20. Expressing enumeration sequence.

5. Correctness of XSDyM to XML grammar language and completeness of the modelling

The set of proposed generic transformation rules described in the previous section facilitates the systematic transformations on the conceptual level of XML modelling into equivalent XSD and Schematron on the logical level.

Each standard for requirements specifications [15] maintain that a set of requirements should be complete. Some recommend ways to achieve completeness that amount to requirements elicitation techniques. However, none gives a way to evaluate completeness or even assess relative completeness. There is research under way to formalize many terms used in software and systems engineering. Ref. [2] formally defines cohesion and coupling using a graph theory. However, there is little information on quantifying the completeness of models, especially at the early conceptual phase.

5.1. Correctness of model transformation

In every model transformation, there is a correlation or correspondence between parts of the input model and parts of the output model which can be specified in terms of abstract semantics and the source and target constructs [26]. Therefore, we adapt the method in Ref.[20] to prove the correctness of our model transformation. The author came out with an approach for the instance-based verification of model transformations based on the concept of structural correspondence. The approach is based on the assumptions that (i) it is relatively easy to specify the correspondence criteria for a transformation, and (ii) the correspondence rules are ‘complete’ in the sense that they cover all the relevant semantic aspects of the transformation.

According to Ref.[20], the transformation can be accepted as correct if a node in the source model and its corresponding node in the target model satisfy some correspondence conditions. The correctness checking is done in four main tasks: (i) identifying the correspondence structure, i.e. of the input model; (ii) specifying the correspondence conditions; (iii) creating the cross links, and (iv) checking the correspondence conditions.

```

<sch:pattern name="Local dynamic constraint">
  <sch:rule context="target">
    <sch:assert test="@name = 'Salary' and newstate >
      oldstate">Salary can never decrease"</sch:assert>
    <sch:assert test= "name = 'MaritalStatus' AND oldstate
      ='single' AND newstate='married'> Not possible</sch:assert>
    <sch:assert test= "name = 'MaritalStatus' AND
      oldstate= 'married' AND newstate=('divorce'|'widow')> Not
      possible</sch:assert>
    <sch:assert test= "name = 'MaritalStatus' AND
      oldstate='divorce' AND newstate='married'> Not possible
    </sch:assert>
  </sch:rule>
</sch:pattern>

```

Fig. 21. Full example of a Schematron schema based on Rule 1 to Rule 8.

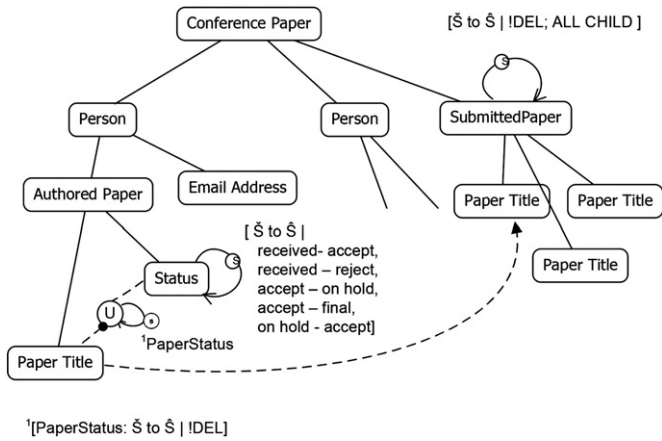


Fig. 22. Corresponding XSDyM model.

5.1.1. Proof of correctness

We show in Fig. 22, the corresponding XSDyM structure and the target model in XSD and Schematron where the figure occupies task (i) of the method. The first task of the method is to identify a sufficient set of node types from the source language that must have a corresponding element in the target model.

Once the pivot nodes have been identified, in task 2, the correctness conditions are specified for each pair of pivot nodes. This task involves traversing the immediate hierarchy of the nodes, access the nodes' attributes and associations. While in task 3, the cross links are used to construct a look-up table (i.e. a map) that matches the corresponding pivot nodes of any instance of a transformation execution. Deploying tasks 2 and 3 produces a look-up table in Table 3 where it shows the cross links of the input model (XSDyM) and the target model (XSD and Schematron).

At the end of the transformation execution, we perform task (iv), i.e. the verification conditions are checked on the instance model. This is

performed by a model checker that performs scanning of the instance models, applying the verification conditions on all the relevant nodes.

We follow the generic model checker provided in Ref. [20] and perform the checking on our set of nodes.

To wrap up the correctness method, Fig. 23 shows the full example of equivalent XSD and the Schematron schema. We convey on the model structure a few related XML constraints but restricted to local dynamic constraints (transformation of non-local constraints will be in the next contribution). This is to verify the model transformation based on our generic transformation rule in the previous section.

5.2. Completeness of the modelling

Completeness refers to wholeness or entirety of an object [14]. Therefore, in this paper, completeness refers to the entirety of a conceptual model.

Based on the work of Ref.[14] we carry out the Completeness Index (CI) as the quality metric to test the capability of our modelling. As per our discussion earlier, completeness means that a conceptual model should contain all true statements about the domain. Based on Ref. [14], CI can be defined as a ratio between the number of constraints represented in a given modelling to the total number of constraints defined for that problem domain. In the mean time, the aim is to get the model to work at least at par with the ORM model in terms of handling the static and dynamic constraint. It is important to mention that in this paper, our modelling approach is focusing on the dynamic constraint than the static ones. We adapt the original metric to suit our problems. Formally, the CI is defined as follows:

Λ is a set of constraints identified from a problem domain, where $\Lambda = \alpha \cup \beta - (\alpha \cap \beta)$, i.e. α is the set of static constraints and β is the set of dynamic constraints.

ξ be the modelling, and f be the projection set of Λ and ξ on Λ , i.e. $f = \phi_{\xi}(\Lambda)$
 $n(\Lambda)$ = total number of constraints in Λ
 $n(\xi)$ = total number of constraints in f ,
 then $CI = n(\xi)/n(\Lambda)$.

Table 3
Lookup table for the cross links.

XSDyM constructs	Graphical notation	Equivalent schema representation
Entity (element)		<xs:element...
Entity (attribute)		<xs:attribute...
Naming convention Leaf element	free style 	example: <xs:element name="SubmittedPaper"> Simple type, <xs:element...
Element with child		<xs:complexType...
Parent-child relation		<xs:complexType... with element declaration
Referencing Non-local constraint transition(single node)		<xs:keyref... <sch:pattern name="..."><sch:rule context="target">...</sch:rule</sch:pattern>
Non-local constraint(single node) – RC	 [EntityName: RC ...]	<sch:assert test="ãI@name=Ebntity Name AND oldstate=..0 newstate=...">Message</sch:assert>
Non-local constraint(cumulative)	 [ObjectName: RC ...]	<sch:pattern name="..."><sch:rule context=EntityName</sch:rule</sch:pattern>
Non-local constraint(cumulative)-RC	 [ObjectName: RC ...]	<sch:assert test=EntityName>Message</sch:assert> or <sch:assert test=EntityName AND EntityName AND ... (with some checking here)>Message</sch:assert>

```

<xs:element name="ConferencePaper" >
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="Person">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="EmailAddress" type="xs:string" minOccurs="0" />
            <xs:element name="AuthoredPaper" minOccurs="0"
maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="PaperTitle" type="xs:string" minOccurs="0" />
                  <xs:element name="Status" type="xs:string" minOccurs="0" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="SubmittedPaper">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="PaperTitle" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:key name="title" >
  <xs:selector xpath="//AuthoredPaper" />
  <xs:field xpath="PaperTitle" />
</xs:key>

<xs:keyref name="titleRef" refer="title">
  <xs:selector xpath="//SubmittedPaper" />
  <xs:field xpath="PaperTitle" />
</xs:keyref>

<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">
  <sch:pattern name="onSingleNode">
    <sch:rule context="target">
      <sch:assert test="@name = 'Status' AND oldstate = 'received'
AND
newstate='accept' "> Not possible</sch:assert>
      <sch:assert test="@name = 'Status' AND oldstate= 'received' AND
newstate='reject' "> Not possible</sch:assert>
      <sch:assert test="@name = 'Status' AND oldstate='accept' AND
newstate= 'on hold' "> Not possible </sch:assert>
      <sch:assert test="@name = 'Status' AND oldstate= 'accept' AND
newstate='final' "> Not possible</sch:assert>
      <sch:assert test="@name = 'Status' AND oldstate='on hold' AND
newstate= 'accept' "> Not possible </sch:assert>
    </sch:rule>
  </sch:pattern>
  <sch:pattern name="SubmittedCollection">
    <sch:rule context="SubmittedPaper">
      <sch:assert test="SubmittedPaper">SubmittedPaper and child cannot be
deleted</sch:assert>
    </sch:rule>
  </sch:pattern>
  <sch:pattern name="PaperStatus">
    <sch:rule context="AuthoredPaper/PaperTitle" role="mandatory">
      <sch:assert test="PaperTitle">PaperTitle cannot be deleted<
      </sch:assert>
    </sch:rule>
    <sch:rule context="AuthoredPaper/EmailAddress">
      <sch:assert test="EmailAddress">EmailAddress cannot be deleted<
      </sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>

```

Fig. 23. Equivalent XSD and Schematron.

Our assumption is α and β exist independently and form totally different sets of constraints, i.e. no β is a generalisation of α , i.e. generalisation of α might require normalization processes. This is to have clear distinction of completeness on both constraints.

To evaluate the completeness of the model, we define the generic requirements of model constraints in the problem domain into few

categories as in Table 4. Assuming that the constraints as per discussion in the previous subtopic are the requirement that needs to be achieved in the modelling, the CI of the model stands on both the static and dynamic constraints. As can be seen, the lines noted with Yes imply that these constraints are comprehended by our approach, while Yes/No implies that these constraints are comprehended but

Table 4
Generic requirements and the coverage of representation using our approach.

Generic modelling requirement		Representation using the modelling	
Structural	Basic	Yes	
	Extensive	Yes/No	
State-based constraints	Static	Yes/No	
	Dynamic	Local	Yes
		Non-local	Yes

with insufficient details, i.e., there are some of the domain requirements that still cannot be expressed using the modelling approach covered in this paper, for example the approach has not yet provided a way to denote explicitly whether a set of nodes comes from a *Complex Type* or a *Simple Type* and also the way to define a specific data-type on a node.

Therefore, based on the table, the result of our CI concluded to be near to optimal in relation to the ability of our approach to interpret the generic requirements into modelling. There is a need to strengthen and establish more features in handling static constraints, e.g., handling more types of integrity constraints, and also the extensive details of the XML structure based on the XSD. However, we believe that our modelling approach is complete to handle the requirements as presented in the earlier part of this paper, and at standard with the ORM in handling dynamic constraint.

6. Discussion

We have proposed a modelling approach for an XML database where it is mainly motivated by the visualisation of dynamic constraints of XML. We believe that our approach could also be used with any semi-structural data of any database. In comparison with the ORM modelling approach, ours is much cleaner and more directed to the nature of XML which is in the forms of a hierarchical model, etc.

Although ORM offers a huge range of graphical elements to convey different constraints, we believe it is less suitable for the XML environment. For instance, ORM is an attribute-free modelling approach, where all the entity types are treated the same, i.e. elements, whereas XML requires the visualisation of attributes. ORM is constructed based on 'roles', which means all relations of the entities must be with the existence of roles which is not necessary for an XML tree model. In addition, in the case of dynamic constraint visualisation, ORM often depends completely on the textual constraint expressions and requires the expression to be noted as footnotes. Nevertheless, ORM is good for a non-structural data model that does not require a hierarchical structure.

Our proposed model is developed based on the natural requirements of XML and offers a clean approach for dealing with dynamic constraints. However, to complete our modelling, we import some elements of ORM into our approach as mentioned in the earlier section. Moreover, using the proposed set of rules, the proposed model can be transformed into equivalent XSD which represents the XML structure and Schematron which represents the dynamic constraints. *W.r.t.* this matter, one of the advantages is that it defines the constraints on top of the existing XSD, i.e. independent of any structural issues.

However, it is worth mentioning that our transformation rules are currently limited to Schematron for expressing dynamic constraints due to a very limited work on dynamic constraint expression for XML. Also, from the set of XSDyM components listed in Table 2, not all are able to be represented using Schematron (at the time of writing) as some are currently still being investigated.

7. Conclusion

This paper presented a graphical modelling notation for XML. We believe that a fit and precise approach is needed to model XML and its constraints. While XML-Schema is widely known as a text-based

grammatical medium for expressing XML constraints i.e. structural constraints, we have to agree that graphical notations are more widely used to visualise the modelling *w.r.t.* designing, reporting, etc., as it provides a more effective way to visualise the data requirements for a human audience. The motivation of our work comes from the problem that the current modelling tool is facing, i.e. to visualise the constraints of XML particularly dynamic constraints.

In developing our own approach, we discovered that the most promising modelling available for dynamic constraints as the time of writing is ORM modelling. ORM offers a range of useful components or notations to convey many kinds of constraints. Unfortunately, we believe that ORM is not the best tool for XML as the connectivity of elements is based on the roles present and it becomes more complex to visualise as the XML becomes bigger.

Our modelling notations provide the simplest way of XML modelling using the tree-structure with the addition of state transition notation inspired from the well-known 'state diagram'. This is specially to support the dynamic constraints of XML where it requires a dynamic transition of two or more points of time. In addition, we retain the use of some ORM components as it offers great usefulness to our model. We show that our model is much more minimal in the context of XML as it does not require the utilization of roles as in the original ORM. We also show in the discussions the compatibility of our model with ORM.

References

- [1] N. Bidoit, S. Cerrito, V. Thion, A first step towards modeling semistructured data in hybrid multimodal logic, *J. Appl. Non Class. Logics* 14 (4) (2004) 447–475.
- [2] L.C. Briand, S. Morasca, V.R. Basilli, Property based software engineering measurement, *IEEE Trans. Softw. Eng.* 2 (1) (1996).
- [3] L. Bird, A. Goodchild, T. Halpin, *Object Role Modelling and XML-Schema*, Springer, Berlin Heidelberg, 2000. 309–322.
- [4] D. Braga, A. Campi, D. Martinenghi, *Efficient Integrity Checking Over XML Documents*, Springer, Berlin Heidelberg, 2006. 206–219.
- [5] H. Chen, H. Liao, A survey to conceptual modeling for XML, 3rd IEEE International Conference on Computer Science and Information Technology, 2010, pp. 473–477.
- [6] M. Curland, T. Halpin, *Enhanced Verbalization of ORM Models*, Springer, Berlin Heidelberg, 2012. 399–408.
- [7] E.O. De Brock, A general treatment of dynamic integrity constraints, *J. Data Knowl. Eng.* 32 (2) (2000) 223–246.
- [8] T. Halpin, *ORM 2*, Springer, Berlin Heidelberg, 2005. 676–687.
- [9] T. Halpin, Business rule modality, *Proc. CAiSE'06 Workshops*, 2006, pp. 383–394.
- [10] T. Halpin, *ORM 2 graphical notation*, Technical report, Newmont University, 2011, (<http://www.orm.net/pdf/ORM2GraphicalNotation.pdf>).
- [11] H. Balsters, T. Halpin, *Formal Semantics of Dynamic Rules in ORM*, Springer, Berlin Heidelberg, 2008. 699–708.
- [12] H. Balsters, A. Carver, T. Halpin, T. Morgan, *Modeling Dynamic Rules in ORM*, Springer, Berlin Heidelberg, 2006. 1201–1210.
- [13] T.A. Henzinger, *Model Checking Game Properties of Multi-agent Systems*, 1443, Springer, Berlin Heidelberg, 1998. 543.
- [14] T. Hussain, S. Shamail, M. Awais, *Schema transformations – a quality perspective*, *Proceedings of 8th IEEE International Multi-topic Conference*, Lahore, Pakistan, 2004.
- [15] P. Kar, M. Bailey, Characteristics of good requirements, *Proceedings of the Sixth Annual International Council on Systems Engineering*, 2, 1996, pp. 284–291.
- [16] T.W. Ling, L.L. Yan, NF-NR: a practical normal form for nested relations, *J. Syst. Integr.* 4 (4) (1994) 309–340.
- [17] L. Linsky, *Reference and Modality*, Oxford University Press, London, 1971.
- [18] Y. Liu, D. Yang, S. Tang, T. Wang, J. Gao, *Extracting Key Value and Checking Structural Constraints for Validating XML Key Constraints*, Springer, Berlin Heidelberg, 2004. 399–408.
- [19] M. Mani, *EReX: A Conceptual Model for XML*, 3186, Springer, Berlin Heidelberg, 2004. 128–142.
- [20] A. Narayanan, G. Karsai, *Specifying the correctness properties of model transformations*, *Proceedings of the Third International Workshop on Graph and Model Transformations*, 2008, pp. 45–52.
- [21] M. Necasky, *XSEM: a conceptual model for XML*, *Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling*, 2007, pp. 37–48.
- [22] F. Neven, *Automata, Logic, and XML*, 2471, Springer, Berlin Heidelberg, 2002. 2–26.
- [23] A. Olivé, *Integrity constraints specification*, Technical Report, LSI, Universitat Politècnica de Catalunya, Barcelona, Spain, 1995.
- [24] G. Psaila, *ERX: a conceptual model for XML documents*, *Proceedings of the 2000 ACM Symposium on Applied Computing*, vol. 2, 2000, pp. 898–903.
- [25] R. Rajugan, E. Chang, L. Feng, T.S. Dillon, *Modeling dynamic properties in the layered view model for XML using XSemantic nets*, *Proceedings of the 2006 International Conference on Advanced Web and Network Technologies, and Applications*, 2006, pp. 42–147.

- [26] A. Sarkar, Conceptual level design of semi-structured database system: graph-semantic based approach, *Int. J. Adv. Comput. Sci. Appl.* 2 (10) (2011) 112–121.
- [27] T. Schwentick, Automata for XML – a survey, *J. Comput. Syst. Sci.* 73 (3) (2007) 289–315.
- [28] A. Sengupta, S. Mohan, R. Doshi, XER – Extensible Entity Relationship Modeling, *Proceedings of the XML 2003 Conference*, 2003, pp. 140–154.
- [29] C.E. Shannon, A mathematical theory of communication, *J. ACM Sigmoblie (Mob. Comput. Commun. Rev.)* 5 (1) (2001) 3–55 (ACM, New York, USA).
- [30] N. Wahid, E. Pardede, Single Transition Constraint for XML Update Validation, *Proc. 15th International Conference on Network-based Information Systems*, 2012, pp. 24–31.
- [31] N. Wahid, E. Pardede, Protecting cumulative node constraint during XML update, *Proc. TrustCom*, 2013, pp. 1795–1802.
- [32] V. Vianu, Dynamic functional dependencies and database aging, *J. ACM (JACM)* 34 (1) (1987) 28–59.
- [33] F. Wenfei, XML constraints: specification, analysis, and applications, *Proc. Sixteenth International Workshop on Database and Expert Systems Applications*, 2005, pp. 805–809.