Front. Comput. Sci., 2015, 9(2): 237–252
DOI 10.1007/s11704-014-3358-5

RESEARCH ARTICLE

A model-driven approach to semi-structured database design

Amir JAHANGARD-RAFSANJANI (✉), Seyed-Hassan MIRIAN-HOSSEINABADI

Department of Computer Engineering, Sharif University of Technology, Tehran 11365-11155, Iran

Abstract Recently XML has become a standard for data representation and the preferred method of encoding structured data for exchange over the Internet. Moreover it is frequently used as a logical format to store structured and semi-structured data in databases. We propose a model-driven and configurable approach for modeling hierarchical XML data using object role modeling (ORM) as a flat conceptual model. First a non-hierarchical conceptual schema of the problem domain is built using ORM and then different hierarchical views of the conceptual schema or parts of it are specified by the designer using transformation rules. A hierarchical modeling notation called H-ORM is proposed to show these hierarchical views and model more complex semi-structured data constructs and constraints. We also propose an algorithm to map hierarchical H-ORM views to XML schema language.

Keywords semi-structured database design, object role modeling, model driven approach

1 Introduction

XML has become a de-facto standard for describing and interchanging semi-structured data among various systems and databases on the Internet. Recently XML has been used by many organizations as a logical format to store data in databases. This increases the need for well-designed XML data models and the need for a methodology for designing XML schemas. Schema definition languages such as XML schema are used for describing valid XML document structures. These languages can be seen as logical models for XML data. They are not suitable to be used at the conceptual level because they force designers to focus on low-level presentation details.

Semi-structured databases and XML have some properties that makes conventional data modeling approaches such as ER and UML unsuitable for designing XML data. Some of these differences are: hierarchical structure, ordering on elements and attributes, mixed content and irregular structures. For example a simple relationship in an ER diagram can be mapped to multiple hierarchical structures.

In this paper we present a model-driven design methodology for XML data using object role modeling (ORM) [1,2] as the platform independent model (PIM). This allows designers to analyze and model data requirements of the problem domain without worrying about later presentations.

We have chosen ORM because its data modeling features are richer than other popular notations such as ER and UML, allowing more business rules to be captured, its attribute free models are more stable than attribute based approaches, because they are free of changes caused by attributes evolving into other constructs or vice versa [3], and its role-based notation allows models to be easily validated with domain experts by verbalization and sample populations [4].

Then we introduce a graphical and hierarchical modeling notation called H-ORM as the platform specific model (PSM) that can be used to show hierarchical views over ORM models. H-ORM supports various semi-structured data properties such as ordering and irregular structures and mixed content. Transformation rules are used by database designers to formally specify the desired hierarchical structure over flat ORM models and generate H-ORM views. Various H-ORM views can be generated over a single ORM model. Then, using a mapping algorithm, an XML schema is generated as the log-

Received October 5, 2013; accepted August 20, 2014

E-mail: jahangard@ce.sharif.edu, hmirian@sharif.edu

ical model of the XML data. Figure 1 shows an overview of our approach to XML database design.

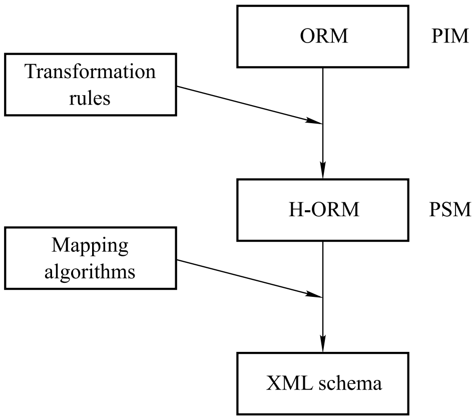

Fig. 1 Model-driven approach to XML data

In this approach the PIM (ORM) is not influenced by XML schema languages and special features of semi-structured data model. The required XML data model representations are modeled in H-ORM in a graphical notation readable to non-technical users. The advantage of H-ORM is that different H-ORM schemes that model different XML representations of the same data, are formally interrelated by the ORM model and transformation rules. This enables to rapidly understand the semantics of the XML schemes and trace the locations of each concept in the XML schemes and vice versa (traceability).

The rest of the paper is organized as follows. Section 2 presents a brief introduction to ORM. In Section 3 we introduce the transformation rules as a means to specify hierarchical views over flat ORM models. Then in Section 4, H-ORM notation is introduced. In Section 5 a mapping algorithm to generate XML schema from H-ORM models is introduced. Various criteria of a conceptual model for XML are presented in Section 6 and our approach is evaluated against these criteria. Related works are studied in Section 7. We conclude the paper in Section 8.

2 Background on ORM

ORM is a conceptual modeling approach that views the world in terms of object-types (entities or values), that play roles in relationships (predicates). It provides graphical and textual languages for verbalizing and querying information as well as various design and transformation procedures and it has semantics based on first-order logic. ORM allows a variety of data constraints to be defined such as mandatory role, subset, uniqueness, exclusion, cardinality and ring constraints.

Figure 2 shows an ORM diagram that models a "conference paper" universe of discourse. In this diagram, object types are represented as named rounded rectangles and relationship-types as named sequences of adjacent role boxes. Individual role names are written in square brackets near each role. A bar over/near a role or role sequence indicates an internal uniqueness constraint, and a circled

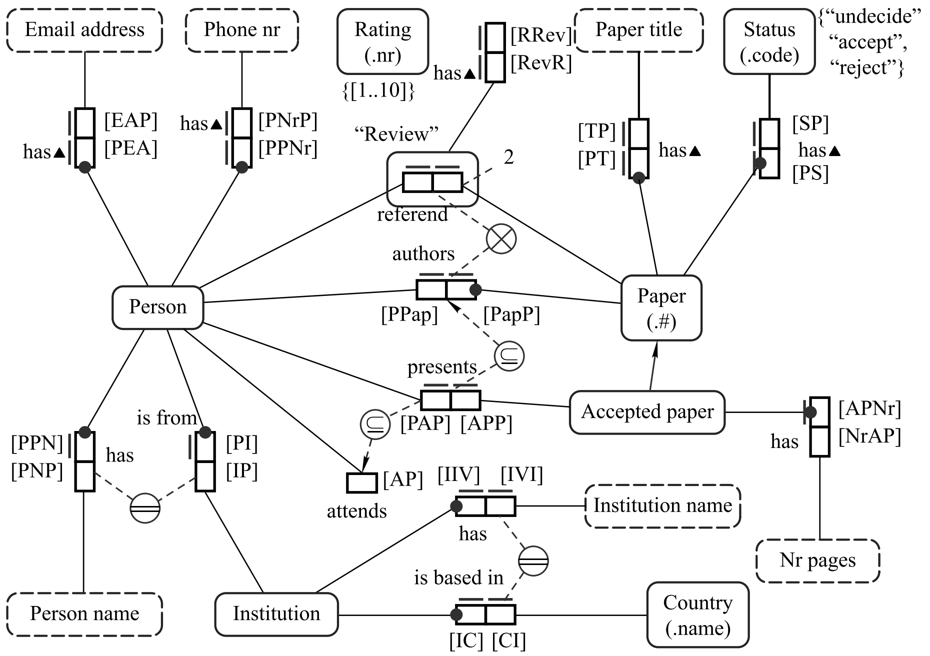

Fig. 2 Conference paper ORM schema

underline or circled double underline denotes an external uniqueness (or primary uniqueness) constraint. For example a *Person* is identified by his/her *Email Address* and an *Institution* is identified by its *name* and *country* combination. Value constraints are represented as a braced list of values, and frequency constraints as a numeric range attached to one or more roles. For example there is a value constraint on *Status* object type which indicates that we have only three *Status* codes: *undecided*, *accepted* and *rejected*; and each *Paper* should be reviewed by exactly two *Persons*. Role and relationship subset constraints are shown as arrow with a circled subset notation, exclusion constraints as a circled "*x*" between the relevant role-sequences, and subtype links as solid arrows between object types. For example an *author* of a *paper* cannot be the *reviewer* of the same *paper* and the presenter(s) of an *accepted paper* should be one (some) of the *authors* of the *paper*.

Figure 3 shows another page of the "conference paper" schema about paper details. The shadow under the *Paper* entity type shows that it is replicated in another page of the schema. A paper may have multiple references and a reference can be from either a journal or a conference proceedings.

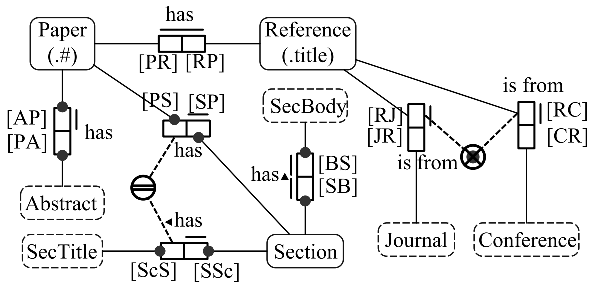

Fig. 3 Conference paper ORM schema-paper details

3 Transformation rules

In order to produce hierarchical representations from flat ORM schemes a designer needs a tool to specify the required hierarchical structure. In this section we introduce transformation rules. These rules are defined on predicates and state how the hierarchy is made. This makes our approach configurable and using various transformation rules a designer can define multiple hierarchical views for special applications over a single ORM schema.

Definition 1 Given predicate R, a transformation rule τ on R is an expression of form:

$$\tau = (R : S(R_S) \rightarrow T(R_T), \langle C \rangle).$$

In which:

- S and T are objects participating in R. S or T can be R itself or its objectified entity type.
- R_S and R_T are roles of S and T in R respectively (if S and T are not R itself). If a player plays one role in the predicate then this parameter can be omitted.
- C is a sequence of form $\langle C_1(R_1), C_2(R_2), \ldots, C_k(R_k) \rangle$ that each C_i is a player of R and R_i is the role played by C_i.
- S is called parent, T is called child and C is called context of the transformation rule.

Let S and T be different from R, at the instance level a transformation rule τ defines a set τ^i of triples (s, t, c) in which $s \in (R_S)^I$, $t \in (R_T)^I$ and c is a sequence $c_1, c_2, \ldots, c_n$, $c_i \in (R_i)^I$ so that:

$$\exists \text{ row} : R^I \cdot (\text{row}.R_S = s \wedge \text{row}.R_T = t \wedge (\forall i : \mathbb{N} | 0 < i \leqslant n \cdot \text{row}.R_i = c_i)).$$

Let S be R and the transformation rule will be $(R : R \rightarrow T(R_T), \langle C \rangle)$ then at the instance level a transformation rule defines a set of triples (r, t, c) in which $r \in R^I$, $t \in (R_T)^I$ and c is a sequence $c_1, c_2, \ldots, c_n$, $c_i \in (R_i)^I$ so that:

$$r.R_T = t \wedge (\forall i : \mathbb{N} | 0 < i \leqslant n \cdot r.R_i = c_i).$$

Let T be R and the transformation rule will be $(R : S(R_S) \rightarrow R, \langle C \rangle)$ then at the instance level a transformation rule defines a set of triples (s, r, c) in which $r \in R^I$, $s \in (R_T)^I$ and c is a sequence $c_1, c_2, \ldots, c_n$, $c_i \in (R_{Ci})^I$ so that:

$$r.R_S = s \wedge (\forall i : \mathbb{N} | 0 < i \leqslant n \cdot r.R_i = c_i).$$

We show the function of the rule by an example. Consider the ternary predicate *Review* shown in Fig. 4(a). *Review* has three player entity types *Person*, *Paper* and *Rating* and models the rating a person gives to a paper. A person can give at most one rating to a paper. Different hierarchical representations of *Review* may be required. For example one may require a list of persons and for each person the list of papers he/she reviews and for each paper the rating given by the person. The hierarchical structure can be specified formally by the following transformation rules:

$$\tau_1 : (\mathit{Review} : \mathit{Person}(r_1) \rightarrow \mathit{Paper}(r_3)),$$
$$\tau_2 : (\mathit{Review} : \mathit{Paper}(r_3) \rightarrow \mathit{Rating}(r_2), \langle \mathit{Person}(r_1) \rangle).$$

These transformation rules define the hierarchical view of the ORM model which is shown in Fig. 4(b). Rule τ_1 is read as "*Person* is the parent of *Paper* in predicate *Review*" and rule τ_2 is read as "*Paper* is parent of *Rating* in the context of *Person* in predicate *Review*".

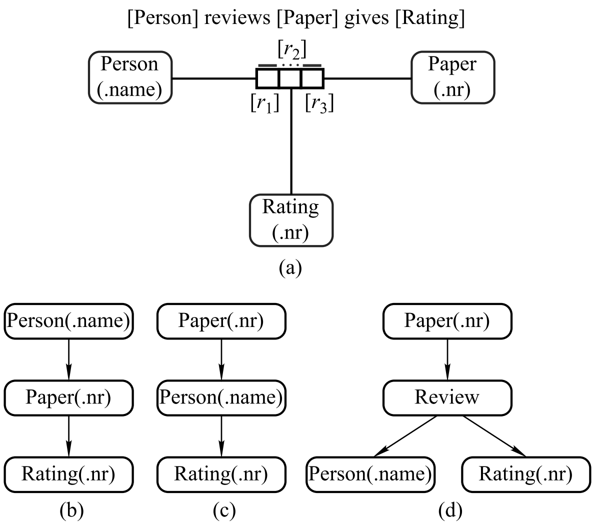

Fig. 4 (a) ORM model for paper review and various; (b), (c), (d) hierarchical views on it

At the instance level τ_1 and τ_2 define the following sets:

$$\tau'_1 = \{(person, paper, \langle\rangle) | \exists r : Review^I \cdot (r.r_1 = person \wedge r.r_3 = paper)\},$$

$$\tau'_2 = \{(paper, rating, \langle person\rangle) | \exists r : Review^I \cdot (r.r_3 = paper \wedge r.r_2 = rating \wedge r.r_1 = person)\}.$$

The above sets organize data in a hierarchical structure in which instances of *Person*, *Paper* and *Rating* are nodes and edges are defined by τ'_1 and τ'_2. Each edge connects parent node to the child node. This hierarchical structure can be easily transformed into XML representations. For example an XML representation defined by τ_1 and τ_2 with sample data is shown in Fig. 5(a).

The transformation rules τ_3 and τ_4 define a hierarchical view of the ORM sample which is shown in Fig. 4(c). At the instance level a structure similar to XML representation of Fig. 5(b) is defined:

$$\tau_3 : (Review : Paper(r_3) \rightarrow Person(r_1)),$$

$$\tau_4 : (Review : Person(r_1) \rightarrow Rating(r_2), \langle Paper(r_3)\rangle).$$

And the transformation rules τ_5, τ_6 and τ_7 specify a hierarchical view of the ORM sample which is shown in Fig. 4(d). At the instance level a structure similar to XML representation of Fig. 5(c) is defined:

$$\tau_5 : (Review : Paper(r_3) \rightarrow Review),$$

$$\tau_6 : (Review : Review \rightarrow Person(r_1), \langle Paper(r_3)\rangle),$$

$$\tau_7 : (Review : Review \rightarrow Rating(r_2), \langle Paper(r_3)\rangle).$$

The transformation rules allow us to specify the required hierarchy but are not usable for specifying more structural requirements of XML data. H-ORM is used to model these specific requirements.

With transformation rules it is possible to specify a hierarchical structure from which the original data cannot be reconstructed. For example the transformation rules:

(a)

```
<Conference>
  <Person name ="Amir">
    <Paper nr="1">
      <Rating nr="3"></Rating>
    </Paper>
    <Paper nr="2">
      <Rating nr="4"></Rating>
    </Paper>
  </Person>
  <Person name="Ali">
    <Paper nr="2">
      <Rating nr="3"></Rating>
    </Paper>
    <Paper nr="1">
      <Rating nr="5"></Rating>
    </Paper>
  </Person>
</Conference>
```

(b)

```
<Conference>
  <Paper nr="1">
    <Person name="Amir">
      <Rating nr="3"></Rating>
    </Person>
    <Person Name="Ali">
      <Rating nr="5"></Rating>
    </Person>
  </Paper>
  <Paper nr="2">
    <Person name="Amir">
      <Rating nr="3"></Rating>
    </Person>
    <Person Name="Ali">
      <Rating nr="4"></Rating>
    </Person>
  </Paper>
</Conference>
```

(c)

```
<Conference>
  <Paper nr="1">
    <Review>
      <Person name="Amir"></Person>
      <Rating nr="3"></Rating>
    </Review>
    <Review>
      <Person Name="Ali"></Person>
      <Rating nr="5"></Rating>
    </Review>
  </Paper>
  <Paper nr="2">
    <Review>
      <Person name="Amir"></Person>
      <Rating nr="3"></Rating>
    </Review>
    <Review>
      <Person Name="Ali"></Person>
      <Rating nr="4"></Rating>
    </Review>
  </Paper>
</Conference>
```

Fig. 5 Various XML representations of ORM schema of Fig. 3

$$(Review : Person(r_1) \rightarrow Paper(r_3)),$$
$$(Review : Person(r_1) \rightarrow Review),$$

specify that for each person there is a list of papers reviewed by the person and there is a list of reviews made by the person. However there is no connection between reviews and papers, i.e., for a given *Review* instance we cannot determine the corresponding *Paper* instance form the specified hierarchical structure. Therefore we introduce some guidelines on specifying transformation rules to prevent these situations.

Suppose a predicate R with players (participants) $O_1, O_2, \ldots, O_n$. Among various possible configurations for specifying transformation rules, two are preferred:

- R is specified as the root. This is specified by transformation rules $(R : R \rightarrow O_1), (R : R \rightarrow O_2), \ldots, (R : R \rightarrow O_n)$. An instance r of R is represented as a root in the hierarchical structure and each $r.O_1, r.O_2, \ldots, r.O_n$ is a child of r, so the original instance of r can be reconstructed from this structure.
- Some of the players of R are specified as parents (ancestors) of R and the others are specified as children of R. For example assume $O_1, O_2, \ldots, O_k$ be ancestors of R and $O_{k+1}, O_{k+2}, \ldots, O_n$ be represented as its children. If we have an instance r of R we can determine $r.O_1, r.O_2, \ldots, r.O_n$ from the corresponding ancestors and $O_{k+1}, O_{k+2}, \ldots, O_n$ can be determined from corresponding children. For example in transformation rules τ_5, τ_6 and τ_7, *Review* is represented as child in τ_5 and is represented as parent in τ_6 and τ_7.
- R is not present in the hierarchy and is replaced by parent-child-context relationship between its participants. For example in transformation rules τ_1 and τ_2, *Review* is not present and is replaced by parent child relationship between *Person*/*Paper* and then parent child relationship between *Paper*/*Rating* in the context of *Person*.

4 H-ORM

H-ORM is a hierarchical data model and H-ORM models can be considered as hierarchical views on ORM schemes. An H-ORM schema is a directed graph. Its nodes are called object types and edges are called relationships. Object types are similar to object types in ORM. An object type in H-ORM represents an object or a predicate from an ORM schema. Relationships represent the nesting of the object types in a hierarchy defined by transformation rules. Formally an H-ORM model derived from and ORM schema $\mathcal{ORM}$ consists of four components:

- A set of nodes (object types) $\mathcal{O}$, representing objects or predicates from $\mathcal{ORM}$.
- A set of directed edges (relationship types) $\mathcal{R}$ representing nesting relationships between the objects.
- A set of constraints $\mathcal{C}$, defined over nodes or edges.
- A reference to the ORM model $\mathcal{ORM}$, on which the H-ORM view is defined.

These concepts are defined formally in Sections 4.1 and 4.2.

4.1 Object types

An object type (a node) in an H-ORM model is a representation of an object or relationship type from $\mathcal{ORM}$. Similar to ORM there are two kinds of object types: *Value types* and *Entity types*. A value type cannot have any edges emanating from it (except in one case described in Section 4.3.6) and it has a predefined data type. Value types are shown by dashed soft rectangles and are representation of value types of the ORM model. An entity type can have edges emanating from it and is shown by a solid soft rectangle. For example in Fig. 6, *Paper*, *Author* and *Section* are entity types and the other object types such as *Title*, *Abstract* and *Name* are value types of type string.

Let $\mathcal{E} \subseteq \mathcal{O}$ and $\mathcal{V} \subset \mathcal{O}$ be the set of entity types and value types, respectively, then $\mathcal{E} \cap \mathcal{V} = \varnothing$ and $\mathcal{E} \cup \mathcal{V} = \mathcal{O}$.

An entity type $e \in \mathcal{E}$ is defined by a triple $(e_{\text{orm}}, e_{\text{label}}, e_{\text{content}})$ in which e_{orm} is a concept (object or predicate) from $\mathcal{ORM}$ and e_{label} is a string called label of the object and e_{content} is the content of the entity which can be a set or a sequence of edges based on ordering on the entity type.

A value type $v \in \mathcal{V}$ is a triple $(v_{\text{orm}}, v_{\text{label}}, v_{\text{type}})$ in which v_{orm} is a value type from $\mathcal{ORM}$, v_{label} is a string called label of the value type and v_{type} is the type of the value type.

4.2 Relationship types

In H-ORM we distinguish among three kinds of relationships: nesting, reference and inheritance. Nesting and reference relationships are direct results of transformation rules. Inheritance relationships are inherited from the corresponding ORM model.

- A nesting relationship type is a triple $(O_{\mathcal{P}}, O_{\mathcal{C}}, \tau)$ in which $O_{\mathcal{P}} \in \mathcal{E}$ is an entity type and $O_C \in O$ is an object type from the H-ORM model called parent and child respectively, and τ is a transformation rule from which the relationship is derived.

- A reference relationship type is a pair $(O_{\mathcal{F}}, O_{\mathcal{R}})$ where $O_{\mathcal{F}} \in \mathcal{E}$ is the referrer entity type and $O_{\mathcal{R}} \in \mathcal{E}$ is the reference entity type. The two entity types should represent the same concept from the ORM model.
- An inheritance relationship type is a pair $(O_{\text{sup}}, O_{\text{sub}})$ where $O_{\text{sup}} \in O$ is the super object type and $O_{\text{sup}} \in O$ is the sub object type.

A nesting relationship is shown by a directed edge from parent to child. A reference relationship is shown by adding the name of the reference entity type with a "↑" to the box of the referrer entity type. For example in Fig. 6, *Author*, *Referee* and *Presenter* entity types refer to *Person* entity type. An inheritance relationship type is shown by a triangle pointing to the super object type on an edge between sub and super object types. If there is not enough space on the diagram or drawing the edge makes the diagram complicated we can add the name of the super object type with a "⊂" symbol to the box of the sub entity type. For example in Fig. 6, *Accepted Paper* is a sub-entity of *Paper* entity-type.

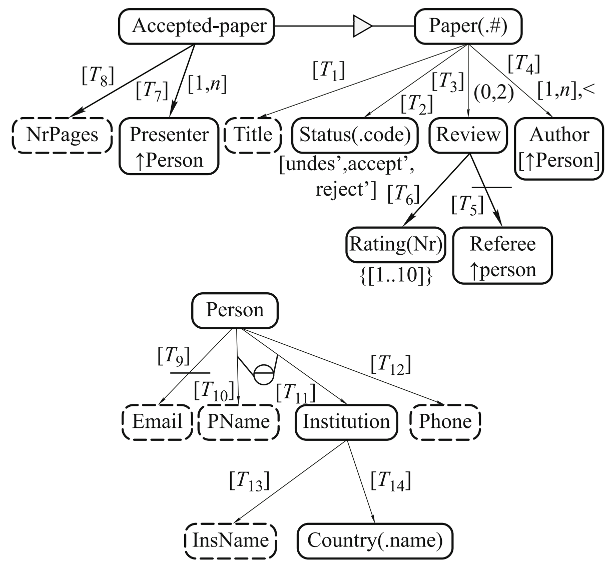

Fig. 6 An H-ORM view over ORM model of Fig. 3

Roles are limited to parent or child, referrer or reference and super-object or sub-object based on the relationship type. Similar to ORM we do not use attributes in order to produce more stable models.

The following transformation rules were used to generate the hierarchical structure of the H-ORM model of Fig. 6. The name of the transformation rule for each edge is written near the edge in the figure.

$T_1 : (hasTitle : Paper \rightarrow Title)$

$T_2 : (hasStatus : Paper \rightarrow Status)$

$T_3 : (refreed : Paper \rightarrow Review)$

$T_4 : (authors : Paper \rightarrow Author(\uparrow Person))$

$T_5 : (hasReview : Review \rightarrow Referee(\uparrow Person))$

$T_6 : (refreed : Review \rightarrow Rating, < Paper >)$

$T_7 : (presents : AcceptedPaper \rightarrow Presenter(\uparrow Person))$

$T_8 : (hasPages : AcceptedPaper \rightarrow NrPages)$

$T_9 : (hasEmail : Person \rightarrow Email)$

$T_{10} : (hasPName : Person \rightarrow Pname)$

$T_{11} : (isFrom : Person \rightarrow Institution)$

$T_{12} : (hasPhone : Person \rightarrow Phone)$

$T_{13} : (hasIName : Institution \rightarrow InsName)$

$T_{14} : (isBasedIn : Institution \rightarrow Country)$

4.3 Additional H-ORM constructs and constraints

The proposed graph model in the previous section is used as backbone of modeling semi-structured data, but in practice more additional and complex constructs and constraints are needed. In this section we introduce various H-ORM constructs and constraints used to model various semi-structured data concepts and irregularities.

4.3.1 Ordering

Various types of ordering can be captured in semi-structured data like XML. For example in XML an ordering on elements is considered. We consider two types of ordering in H-ORM:

- Ordering on a set of object type instances related to another object type instance, e.g., ordering on authors of a paper. We simply say that the edge/relationship between two object types is ordered and denote it by a "<" symbol on the edge. For example in the H-ORM view of Fig. 6, there is an ordering on the edge between *Paper* and *Author* Object types. This indicates that there is an ordering on the authors of a paper which should be preserved.
- Ordering on edges emanating from an entity type (its content), e.g., ordering on parts of a paper such as title, authors, abstract and chapters. We simply say that an object type is ordered and show this by a "<" symbol after the label of the object type. If an entity type $e \in \mathcal{E}$ is

ordered then e_{content} is a sequence of edges, otherwise it is a set of edges. This ordering must be preserved when translating this object type to other representations such as XML schema.

The H-ORM view in Fig. 7 contains three ordered object types, namely, *Paper*, *Author* and *Section*. For example, ordering on Section models that for each section first comes the *SecTitle* and then the one or more paragraphs.

4.3.2 Participation

A participation constraint can be defined on a nesting relationship. It specifies the number of instances of child object type that may be nested in a single instance of the parent object type. This constraint is of the form (min, max) where min and max are the minimum and maximum participation of the child object type in relationship, respectively. The default participation is $(1, 1)$. The usual shorthand can also be used to represent the participation constraints, "?" represents $(0, 1)$, "$*$" represents $(0, n)$ and "+" represents $(1, n)$. A minimum value larger than zero in a participation constraint, indicates a mandatory relationship between parent and child.

The participation constraints are derived from the ORM model. If the parent object type plays an optional role in the predicate of the transformation rule of the edge then the minimum is "0", and if there is a multi-role uniqueness constraint on the predicate of the transformation rule then the maximum will be "*n*".

For example in H-ORM view of Fig. 6, there is an implicit $(1, 1)$ participation constraint between *Paper* and *Title* meaning that each paper has exactly one title and each title belongs to one paper. The $(1, n)$ participation constraint between *Paper* and *Author* indicates that a paper has one or more than one authors.

4.3.3 Choice

Choices are used to model irregular structures in semi-structured data. A choice constraint can be defined on a set of relationship types having a single parent. In this case an instance of the parent object can only participate in one of the relationships. A choice construct may have participation constraint to represent the minimum and maximum occurrence of the entire choice. In H-ORM a choice is represented by a diamond with a vertical bar. The direction of the edges is from parent to the diamond and from the diamond to the child object sets. The default participation is $(1, 1)$.

For example in Fig. 7, there is a choice constraint on edges emanating from *Reference* entity type meaning that a reference can be from a *Conference* or a *Journal* but not from both at the same time. This is a representation of the exclusive-or constraint on Reference roles from ORM schema of Fig. 3.

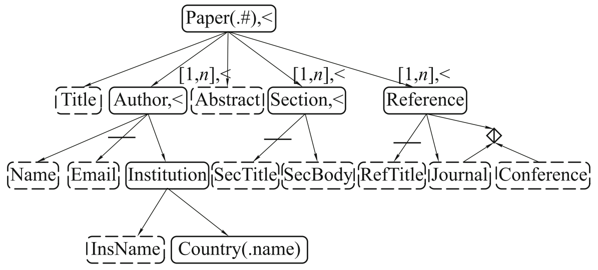

Fig. 7 H-ORM view over ORM model of Fig. 3

4.3.4 Value

A value constraint can be defined on a value type and it determines the domain of the object instances. A value constraint may be defined by declaring its set of possible values as one or more enumerations or ranges enclosed in braces (curly brackets). In an H-ORM diagram, a value constraint on a value type is declared by displaying the extension next to the value type. Value constraints are inherited from ORM schema.

If we list or enumerate all the possible values, this is an *enumeration*, If the values may be ordered in a continuous list from first to last, we can simply list the first and last with ".." between, since we know how to fill in the intermediate values. This is called a range. If a range is unbounded at one end, no value appears at that end.

For example in Fig. 6, there is an enumeration value constraint on values of *Status* code which restricts its values to "undecided", "accept" and "reject"; and there is a range value constraint on *RateNo* values restricting its values to a range between 1 and 10 inclusive.

4.3.5 Uniqueness constraints

Because H-ORM models are hierarchical, uniqueness constraints are defined in the context (scope) of the entity type.

Two kinds of uniqueness constraints can be defined in H-ORM:

- An entity type can have a reference scheme which is written in parentheses near its label. For example in each *Paper* has a paper number which is unique within paper context; and because in Fig. 6 and Fig. 7, *Paper* is a root node then paper number is unique globally within the matching XML document.

- For some entity types it is needed to use its content (children) to define keys. In this case the key is shown by a line on the edge connecting a parent to its key child. If the key is compound then a notation similar to ORM external uniqueness constraint (a circle with a horizontal line inside it connecting key components) is used to show key components. For example in the H-ORM view of Fig. 6, *Email* is unique for *Person* meaning that there are no two persons having the same email; But in the H-ORM view of Fig. 7 *Email* is unique for Author in the scope of *Paper* meaning that no two authors of a paper can have similar email addresses.

Key constraints can be derived from uniqueness constraints which play the identifier role of the object types in ORM model. If there is more than one uniqueness constraint on a single entity type then one of them is chosen as the primary reference scheme (key) and is shown by a double line instead of a single line. For example in Fig. 6, the combination of person name and institution is unique and is the primary key of the person.

4.3.6 Mixed content

Another useful feature that a conceptual model should be able to provide for semi-structured data is the ability to represent mixed content. For example XML schema provides a *mixed* attribute that can be set to true. Setting *mixed* to true enables character data to appear between the child elements of the mixed element in a compatible XML document. In order to model mixed content in H-ORM we use a value type for the object type with mixed content. In this case the value type can have edges emanating from it. In H-ORM we do not explicitly specify how text in mixed with child elements. If the pattern matters the designer must model nodes explicitly rather than using the generic mixed type.

Formally a mixed type is like creating a relationship to a value type of type string. The string associated with a mixed object instance may be scattered among direct child object types.

5 Mapping H-ORM to XML schema

Having modeled the desired hierarchical structure of the XML data in an H-ORM model, in this section we provide the mapping algorithm from H-ORM to the XML schema. In translation from H-ORM to XML schema we should consider the following challenging issues:

- Translation to valid XML-schema instance sometimes need extra artifacts to satisfy XML schema's requirements (e.g., a sequence among concepts when the model has no requirement for a sequence).
- In some cases multiple translations are possible. In our mapping algorithm we point out alternatives and we propose a default translation.
- There are some features in ORM that are not available in XML schema, e.g., multiple inheritance, disjunctive mandatory constraints, ring constraints. These limitations should be considered during translation and are discussed in the algorithm.

The mapping algorithm consists of three steps which are described in the following three sub-sections.

5.1 Generating simple type definition for H-ORM value types

For a value type we can easily map it to a built-in simple type in XML schema. XML schema has a rich built-in type system. For example *Email* is mapped to:

```
<xs:simpleType name="Email">
    <xs:restriction base="xs:string"/>
< /xs:simpleType>
```

In order to enforce value constraints we can use the "restriction" mechanism for type creation provided by XML schema. We can derive a new simple type by restricting an existing simple type through various restrictions. For example inclusive range restrictions can be expressed by "minInclusive" and "maxInclusive" and enumeration value constraints can be implemented using the "enumeration" restriction.

For example *RateNr* value type from Fig. 6 which has a range value constraint {[1..10]} is mapped to:

```
<xs:simpleType name="RateNr">
   <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="10"/>
    < /xs:restriction>
< /xs:simpleType>
```

and *StatusCode* is mapped to:

```
<xs:simpleType name="StatusCode">
   <xs:restriction base="xs:string">
      <xs:enumeration value="undec"/>
      <xs:enumeration value="accept"/>
      <xs:enumeration value="reject"/>
   < /xs:restriction>
< /xs:simpleType>
```

5.2 Generating complex type definitions for H-ORM entity types

In mapping H-ORM entity types to XML schema complex type definitions we distinguish between three kinds of entity types:

- **Referrer**: An entity type that refers to another entity type, i.e., is the referrer object type in a reference relationship type. For example, *Presenter* and *Author* refer to *Person* in Fig. 6.
- **Sub-entity**: An entity type that is the sub object type in an inheritance relationship type such as *AcceptedPaper* which is a sub-type of *Paper* in Fig. 6.
- **Normal**: An entity type that is neither referrer nor Inherited.

For a normal entity type we create a complex type in XML schema and we represent all its content as XML schema sub-elements and attributes.

In order to map content of the H-ORM entity types sub-elements and attributes are used. If a child of the entity type is an entity type then it is mapped to a sub-element and if it is a value type it is mapped to a sub-element or an attribute based on other constraints.

For a referrer entity type, a complex type is created and only elements and attributes that construct the primary reference scheme (primary key) of the referenced entity type is added as sub-elements and attributes. If an entity type has an inline primary key it will be mapped to an attribute.

For a sub-entity type a complex type is created and only elements and attributes that construct the primary reference scheme (primary key) of the super-entity type is added as sub-elements and attributes.

Various factors have influence on mapping the content of an entity type here we discuss these factors and describe how to map various H-ORM constraint and structures:

5.2.1 Order

If the content of an entity type is ordered, it is mapped using XML schema *sequence* structure, and if the order is not important, it is mapped using XML schema *all* structure. Because of various restrictions in XML schema using these structures is not straightforward.

Using *all* structures in XML schema imposes various (likely unwanted) restrictions. For example it allows its children to appear at most one time within the *all* structure and secondly using extension mechanism is not possible for two elements if *all* structure is used in any of them.

Only in the one case we will use *all* structure to map an unordered entity type: when all the child types in the content of the entity type have maximum participation of "1".

Another solution for the cases in which participation of a child type is more than "1" is to introduce a *container* element. The child type is defined in the content of the container element instead of being defined in the content of the parent type and the container element is added to the content of the parent type with "maxOccures=1". The name of the container element is generated by pluralizing the name of the child element (adding "s" to its name).

For example two possible ways to map *Paper* contents (see Fig. 6) is shown in Fig. 8. In Fig. 8(a) its content is mapped using *container* elements and *all* structure. For example in line 3 a container element is introduced for *Review* element

```
<xs:complexType name="Paper">
  <xs:all>
    <xs:element name="Reviews" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Review" type="Review"
minOccurs="0" maxOccurs="2"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Authors">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Author" type="Author"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:all>
  <xs:attribute name="Title" type="Title"
use="required"/>
  <xs:attribute name="PaperNo" type="PaperNo"/>
</xs:complexType>
```

(a)

```
<xs:complexType name="Paper">
  <xs:sequence>
    <xs:element name="Review"
        type="Review" minOccurs="0"
        maxOccurs="2"/>
    <xs:element name="Author"
   type="Author" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Title" type="Title"
        use="required"/>
  <xs:attribute name="PaperNo"
        type="PaperNo"/>
</xs:complexType>
```

(b)

Fig. 8 Two options in mapping entity type contents. (a) using container elements and all structure; (b) using sequence structure

named *Reviews* because *Review* has “maxOccurs=2” and in line 10 a container element named *Authors* is introduced for *Author* element because it has “maxOccurs=unbounded”. In Fig. 8(b) a sequence structure is used instead. This makes schema simpler but it introduces an unwanted order on sub-elements in the corresponding XML document.

5.2.2 Participation

For sub-elements, participation constraints are mapped to “minOccurs” and “maxOccurs” attributes. Because sub-elements are mandatory and functional there is no need to explicitly state (1, 1) participation constraint for them. If the maximum participation is “*n*” then the “maxOccurs=unbounded” is used.

For example in Fig. 8(b) line 3, *Review* entity is mapped to a sub-element in the content of *Paper* having “minOccurs=0” and “maxOccurs=2” meaning that each paper can have at most two reviews. In line 4 of Fig. 8(b), *Author* is mapped to a sub-element in the content of *Paper* having “maxOccurs= unbounded” which means that a paper can have unlimited number of authors.

If a value type child has a max participation of “1” and the content of the parent type is not ordered, it can be mapped to an attribute. Attributes are by default optional in XML schema so if it is mandatory in H-ORM it should be explicitly stated by a “use=required” attribute.

For example in Fig. 8(b) line 8 an attribute is introduced for *Title* value type; and because its minimum participation is “1”, a “use=required” attribute is defined on it.

5.2.3 Choice

Choices in content of an entity type will be mapped to XML schema choice structure. If there is a choice constraint in the content of an entity type, the only choice is to map its content is XML schema sequence structure even if the entity type is not ordered because XML schema does not allow choices inside *all* structure.

For example the *Reference* entity type from Fig. 7 is mapped to:

```
<xs:complexType name="Reference">
   <xs:sequence>
      <xs:element name="RefTitle"
              type="RefTitle"/>
      <xs:choice>
         <xs:element name="Journal"
              type="Journal"/>
         <xs:element name="Conference"
              type="Conference"/>
      < /xs:choice>
   < /xs:sequence>
< /xs:complexType>
```

5.3 Creating root element and defining keys and references

Each XML schema instance must have a single root element so we create a root element to represent the whole H-ORM view. A sub-element will be created for each root entity type in H-ORM view. For example the root element definition for H-ORM view of Fig. 6 is:

```
<xs:element name="ConferencePapers">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Paper" type="Paper"
              maxOccurs="unbounded"/>
      <xs:element name="Person" type="Person"
              maxOccurs="unbounded"/>
      <xs:element name="AcceptedPaper"
              type="AcceptedPaper"
              maxOccurs="unbounded"/>
    < /xs:sequence>
  < /xs:complexType>
< /xs:element>
```

Then the primary key of root entity types will be mapped to XML schema keys. Because the root element is the only element definition in our mapping algorithm all key, keyref and uniqueness constraints are defined in this level. For example the primary key of *Person* from Fig. 6 is mapped to:

```
<xs:key name="PersonKey">
   <xs:selector xpath="Person"/>
   <xs:field xpath="@PName"/>
   <xs:field xpath="Institution/InsName"/>
   <xs:field
      xpath="Institution/Country/CName"/>
< /xs:key>
```

Uniqueness constraints on non-root entity types and secondary uniqueness constraints on entity types are mapped to XML schema *unique* definitions. For example uniqueness constraint on Emails in Fig. 6 is mapped to:

```
<xs:unique name="EmailUnique">
   <xs:selector xpath="Person"/>
   <xs:field xpath="@Email"/>
< /xs:unique>
```

The uniqueness constraint defined on *Referee* sub-entity of *Review* entity in the scope of a *Paper*, meaning that a review can be done by a single reviewer on a single paper, is mapped to the following unique definition in the definition of *Paper* element:

```
<xs:unique name="ReviewUnique">
```

```
   <xs:selector xpath="Review"/>
   <xs:field xpath="Referee/@PName"/>
   <xs:field
      path="Referee/Institution/InsName"/>
   <xs:field
   xpath="Referee/Institution/Country/CName"/>
< /xs:unique>
```

Reference and inheritance relationship types are mapped to *KeyRef* definitions in XML schema. For example the reference between Presenter and Person in Fig. 6 is mapped to:

```
<xs:keyref name="PresenterPersonRef"
    refer="PersonKey">
  <xs:selector
    xpath="AcceptedPaper/Presenter"/>
  <xs:field xpath="@PName"/>
  <xs:field xpath="Institution/InsName"/>
  <xs:field
    xpath="Institution/Country/CName"/>
< /xs:keyref>
```

Table 1 summarizes the mapping between H-ORM concepts and XML schema constructs done by the transformation algorithm. Having multiple options on the right-hand side indicates various choices on mapping an H-ORM concept.

Table 1 H-ORM concepts and the resulting XML-schema constructs

H-ORM concept	XML-schema
Value type	Simple type/Element/Attribute
Value constraints	Restriction mechanism
Normal entity type	Complex type (Full content model)
Referrer entity type	Complex type (Key content model)
Sub-entity type	Complex type (Key content model)
Ordered content	Sequence
Unordered content	All/Sequence/Container Element
Participation constraints	minOccurs/maxOccurs
Choice	Choice
Primary key/uniqueness constraints	Key/Unique
Reference/inheritance relationships	KeyRef

For the complete XML schema definition generated from the H-ORM schemas of Fig. 6 and Fig. 7, please refer to Appendix A.

6 Evaluation

For evaluating conceptual modeling languages in general, the following criteria are introduced in [5]: *expressibility*, *clarity*, *simplicity* and *orthogonality*, *semantic stability*, *semantic relevance*, *validation mechanisms*, *abstraction mechanisms*, and *formal foundation*. In Chapter 3 of [5] ORM is compared with ER and UML with these criteria and since we use ORM as the platform independent model, the same discussion is applicable here.

In [6,7] lists of requirements for conceptual models for XML are presented. Some of these requirements cover general goals of the XML conceptual modeling and the others are specific to XML modeling constructs. Here we evaluate our approach with these criteria for XML conceptual models:

- Graphical Notation We have presented a graphical notation called H-ORM which will be used along ORM.
- Formal Foundation ORM has a solid formal foundation in first-order logic[5]. Transformation rules and H-ORM constructs and constraints are formally specified.
- Structure Independence By choosing a model-driven approach the designer can model hierarchical aspect of XML in PSM (H-ORM) but the PIM (ORM) is independent from these structural properties.
- Logical Level Mapping There should be algorithms for mapping of the conceptual modeling constructs to the XML logical level. We presented a mapping algorithm to map H-ORM views to XML schema.
- Views It should be possible to present different hierarchical views of the same data. A designer can define various H-ORM views over a single ORM model.
- Ordering H-ORM supports two types of ordering. Ordering on a set of object type instances related to another object type instance and ordering on the content of an entity type.
- *N*-ary Relationship Types ORM supports *N*-ary relationship types.
- Irregular and Heterogeneous Structure Irregular structure is modeled using various conceptual constructs specially exclusion constraint in ORM and choice in H-ORM.
- Cardinality for All Participants In addition to the cardinalities defined in ORM, cardinality constraints can be defined on parent and child relationships in the H-ORM model.
- Document-centric Data In order to model mixed content in H-ORM we use a value type for the object type with mixed content.

7 Related works

There are various approaches to model XML data. At the logical level schema languages such as XML schema and Relax NG are used to specify XML structures. Several commercial tools provide support for graphically representing XML schema structures. For example XML spy[1)] and stylus studio[2)] have their own proprietary notations to represent XML structures graphically. These products provide a graphical visualization of XML schema but they do not provide a higher level of abstraction.

Another approach to model XML data at the conceptual level is using existing modeling methods such as ER and UML. But XML data has some features that are not easily captured in traditional conceptual methods. These features include: ordering on concepts, irregular structure, hierarchical nature and mixed content. For example ER does not support views, ordering, irregular and heterogeneous structure and document centric data criteria for XML conceptual models. So in order to use the traditional conceptual modeling methods researchers had to extend them. For example Badia in [8] proposed a minimal extension to ER to make it suitable for DTD. The author suggested choice attribute to model choice between attributes and marking attributes as required or optional. The proposed method does not support hierarchical structures explicitly and modeling of ordering and document-centric data is not possible. Mani in [9] proposed an extension to ER called EReX by introducing ordering constraints on participants of relationship types, coverage constraints on entity types and roles. Again the author does not introduce any special kind of relationship type to model hierarchical structures and modeling of document-centric data is not possible too and ordering is only possible on the participants of the relationship type. Sengupta et al. [10] proposed another extension to ER called XER by adding ordering on entities and mixed entities. In XER only binary relationship types without attributes and with the cardinality type $1 : N$ is supported. Modeling of irregular structure is not supported. Psaila in [11] introduced ERX, extending ER by binary relationship types with alternatives, containment relationship type, ordering attributes and interfaces. The author does not propose any algorithm for mapping ERX schema to logical level in the paper and only binary relationship types without attributes are supported. Necaský in [12] proposed XSEM in which ER is extended by adding order on attributes of entity and relationship types, data node types, outgoing and incoming cluster types. Then a method to define various hierarchical views on the model is proposed.

In UML-based approaches such as [13,14] usually a UML profile that is composed of UML stereotypes for modeling constructs of a certain XML schema language is proposed.

Dobbie et al. in [15] propose a hierarchical modeling language called ORA-SS that is especially designed for semi-structured data. Using this approach one can model hierarchical structures easily but a problem for non-hierarchical structures arises. Furthermore, the hierarchical nature of this approach forces designers to make decisions about the hierarchical structure of the model too early in the modeling process. Al-Kamha et al. in [16] propose C-XML by extending OSM [17] adding choice, sequence, mixed content and general co-occurrence constraints. C-XML does not provide concepts to model hierarchical structure, document centric data and irregular data explicitly.

There are comprehensive surveys on conceptual modeling for XML such as [6,7,18] that many details about above works and their strengths and weaknesses are discussed.

In case of using ORM as a conceptual model for XML to our knowledge only one work exists in which Bird et al. [19] propose an algorithm to automatically generate an XML-schema for an ORM conceptual model. Their goal is to reduce redundancy and increase connectivity in the resulting XML instances. But their approach is not configurable and the designer has no control over the generated schema.

8 Conclusion and future works

In this paper we introduced a model-driven and configurable approach to XML database design. In our approach we used ORM as the platform independent model and we introduced H-ORM as the platform specific model for XML data. Then we introduced an algorithm to map H-ORM models to XML-schema. By dividing the design process to two levels the designer do not need to focus on hierarchical structure of the data in early stages of the design process. We are now working on implementing our approach in NORMA [20]. NORMA is a free and open source plugin for Visual Studio. Currently the tool supports entry of ORM2 schemas, verbalization of most constraints and code generation to a variety of database management systems. We are extending the tool by adding a “Transformation Rule Editor” window in which the designer can enter transformation rules and then the H-ORM schema is generated based on the transformation rules. We also are adding additional H-ORM concepts to the ORM designer toolbox in order to allow the designer to further specify H-ORM constraints on the generated model. Figure 9 sum-

1) XML spy. http://www.altova.com/xmlspy.html: Altova
2) Stylus studio. http://www.stylusstudio.com/xml_schema_editor.html: Progress Software Corporation

marizes the main components of NORMA. The components that we are adding to the tool are shown in dashed rectangles.

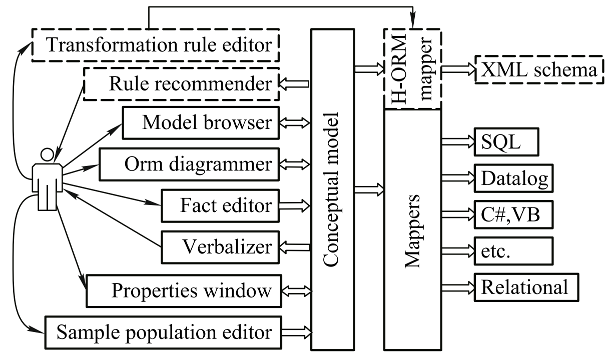

Fig. 9 Main components of NORMA

Another feature that we are currently working on, is transformation rule suggestion. For example by considering reducing redundancy as a goal we can suggest a transformation rule set on a given ORM model to the database designer. We are working on these suggestions in our implementation and it will be included in the final tool.

Recently the new edition of XML schema called XML schema 1.1 [21] has been introduced which has more flexibility than XML schema 1.0 [22]. Among the new features, three of them are more promising to help us in producing better mapping algorithms:

- **Assertions** allow us to specify constraints using XPath 2.0 expressions. Using assertions can help us in mapping more ORM and H-ORM constraints to XML schema. This can help us to better map some complex business rules defined in ORM model that where not possible to specify in XML schema 1.0.
- **Type Alternatives** allow a type to be dynamically assigned to an element based on the values of its attributes. This can help us to better map constraints like exclusion constraints and choices.
- **All Group** The all element allows elements with multiple occurrence. This can help us to better map unordered entities to XML schema.

We are considering these features to enhance our mapping algorithm to include more constraints and produce a better XML schema.

Appendix Generated XML schema from algorithm

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ConferencePapers">
    <xs:complexType><xs:sequence>
        <xs:element name="Paper" type="Paper" maxOccurs="unbounded">
          <xs:unique name="ReviewUnique"><xs:selector xpath="Review"/>
            <xs:field xpath="Referee/@PName"/>
            <xs:field xpath="Referee/Institution/InsName"/>
            <xs:field xpath="Referee/Institution/Country/CName"/>
          </xs:unique>
        </xs:element>
        <xs:element name="Person" type="Person" maxOccurs="unbounded"/>
        <xs:element name="AcceptedPaper" type="AcceptedPaper" maxOccurs="unbounded">
          <xs:unique name="PresenterUnique">
            <xs:selector xpath="Presenter"/><xs:field xpath="@PName"/>
            <xs:field xpath="Institution/InsName"/>
            <xs:field xpath="Institution/Country/CName"/>
          </xs:unique>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:key name="PaperKey"><xs:selector xpath="Paper"/><xs:field
           xpath="@PaperNo"/></xs:key>
    <xs:keyref name="AcceptetPaperRef" refer="PaperKey">
      <xs:selector xpath="AcceptedPaper"/><xs:field xpath="@PaperNo"/>
    </xs:keyref>
    <xs:key name="AcceptedPaperUnique">
      <xs:selector xpath="AcceptedPaper"/><xs:field xpath="@PaperNo"/>
    </xs:key>
    <xs:key name="PersonKey">
      <xs:selector xpath="Person"/>
      <xs:field xpath="@PName"/><xs:field xpath="Institution/InsName"/>
      <xs:field xpath="Institution/Country/CName"/>
    </xs:key>
    <xs:unique name="EmailUnique"><xs:selector xpath="Person"/><xs:field xpath="@Email"/>
    </xs:unique>
    <xs:keyref name="PresenterPersonRef" refer="PersonKey">
      <xs:selector xpath="AcceptedPaper/Presenter"/>
      <xs:field xpath="@PName"/><xs:field xpath="Institution/InsName"/>
      <xs:field xpath="Institution/Country/CName"/>
    </xs:keyref>
  </xs:element>
  <xs:complexType name="Paper2"><xs:all>
      <xs:element name="Reviews" minOccurs="0"><xs:complexType>
          <xs:sequence><xs:element name="Review" type="Review" minOccurs="0" maxOccurs="2"/>
          </xs:sequence></xs:complexType>
      </xs:element>
      <xs:element name="Authors"><xs:complexType>
          <xs:sequence><xs:element name="Author" type="Author" maxOccurs="unbounded"/>
          </xs:sequence></xs:complexType>
      </xs:element>
    </xs:all>
    <xs:attribute name="Title" type="Title" use="required"/>
    <xs:attribute name="PaperNo" type="PaperNo"/>
  </xs:complexType>
  <xs:complexType name="Paper"><xs:sequence>
      <xs:element name="Review" type="Review" minOccurs="0" maxOccurs="2"/>
      <xs:element name="Author" type="Author" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Title" type="Title" use="required"/>
    <xs:attribute name="PaperNo" type="PaperNo"/>
  </xs:complexType>
  <xs:complexType name="Person">
    <xs:all><xs:element name="Institution" type="Institution"/></xs:all>
    <xs:attribute name="PName" type="PName" use="required"/>
    <xs:attribute name="Email" type="Email" use="required"/>
    <xs:attribute name="Phone" type="Phone"/></xs:complexType>
  <xs:complexType name="Rating">
    <xs:sequence><xs:element name="RateNo" type="RateNo"/></xs:sequence>
  </xs:complexType>
  <xs:complexType name="Review"><xs:all>
      <xs:element name="Referee" type="Referee"/><xs:element name="Rating" type="Rating"/>
    </xs:all></xs:complexType>
  <xs:complexType name="Institution"><xs:all>
      <xs:element name="InsName" type="InsName"/><xs:element name="Country" type="Country"/>
    </xs:all></xs:complexType>
  <xs:complexType name="Country">
    <xs:sequence><xs:element name="CName" type="CName"/></xs:sequence>  </xs:complexType>
  <xs:complexType name="AcceptedPaper">
    <xs:sequence><xs:element name="Presenter" type="Presenter" minOccurs="0"
maxOccurs="unbounded"/></xs:sequence>
    <xs:attribute name="PaperNo" type="PaperNo"/>
  </xs:complexType>
  <xs:simpleType name="RateNo">
    <xs:restriction base="xs:byte">
```

```
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="10"/>
    < /xs:restriction>
  < /xs:simpleType>
  <xs:simpleType name="Email"><xs:restriction base="xs:string"/></xs:simpleType>
  <xs:simpleType name="Phone"><xs:restriction base="xs:string"/></xs:simpleType>
  <xs:simpleType name="PName"><xs:restriction base="xs:string"/></xs:simpleType>
  <xs:simpleType name="PaperNo"><xs:restriction base="xs:int"/></xs:simpleType>
  <xs:simpleType name="PageNr"><xs:restriction base="xs:int"/></xs:simpleType>
  <xs:simpleType name="Title"><xs:restriction base="xs:string"/></xs:simpleType>
  <xs:simpleType name="InsName"><xs:restriction base="xs:string"/></xs:simpleType>
  <xs:simpleType name="CName"><xs:restriction base="xs:string"/></xs:simpleType>
  <xs:complexType name="Author">
    <xs:all><xs:element name="Institution" type="Institution"/></xs:all>
    <xs:attribute name="PName" type="PName" use="required"></xs:attribute>
  < /xs:complexType>
  <xs:complexType name="Presenter">
    <xs:all><xs:element name="Institution" type="Institution"/></xs:all>
    <xs:attribute name="PName" type="PName" use="required"/></xs:complexType>
  <xs:complexType name="Referee">
    <xs:all><xs:element name="Institution" type="Institution"/></xs:all>
    <xs:attribute name="PName" type="PName" use="required"/></xs:complexType>
< /xs:schema>
```

References

1. Halpin T. A logical analysis of information systems: static aspects of the data-oriented perspective. PhD dissertation. University of Queensland, 1989
2. Halpin T A, Morgan A J, Morgan T. Information modeling and relational databases. Morgan Kaufmann, 2008
3. Halpin T, Bloesch A. Data modeling in UML and ORM: a comparison. IDEA Group Publishing Company, 1999, 4–13
4. Bloesch A C, Halpin T A. Conceptual queries using ConQuer-II. Conceptual modeling—ER'97. Springer, 1997, 113–126
5. Halpin T, Morgan T. Information modeling and relational databases. Morgan Kaufmann Publishers Inc., 2008
6. Sengupta A, Wilde E. The case for conceptual modeling for XML. TIK Report 244. 2006
7. Necaský M. Conceptual modeling for XML: a survey. In: Proceedings of the DATESO 2006 Annual International Workshop on Databases, Texts, Specifications and Objects (DATESO 2006). 2006, 40–53
8. Badia A. Conceptual modeling for semistructured data. In: Proceedings of the 3rd International Conference on Web Information Systems Engineering (Workshops) (WISEw'02). 2002, 170–177
9. Mani M. EReX: a conceptual model for XML. In: Bellahsène Z, Milo T, Rys M, Suciu D, Unland R, eds. Database and XML technologies. Springer Berlin Heidelberg, 2004, 128-142
10. Sengupta A, Mohan S, Doshi R. XER-extensible entity relationship modeling. In: Proceedings of the XML 2003 Conference. 2003, 140–154
11. Psaila G. ERX: a conceptual model for XML documents. In: Proceedings of the 2000 ACM Symposium on Applied Computing, Volume 2. 2000, 898–903
12. Necaský M. XSEM: a conceptual model for XML. In: Proceedings of the 4th Asia-Pacific Conference on Comceptual Modelling, Volume 67. 2007, 37–48
13. Narayanan K, Ramaswamy S. Specifications for mapping UML models to XML schemas. In: Proceedings of the 4th Workshop in Software Model Engineering. Montego Bay, Jamaica, 2005
14. Routledge N, Bird L, Goodchild A. UML and XML schema. In: Proceedings of the 13th Australasian Database Conference, Volume 5. 2002, 157–166
15. Dobbie G, Xiaoying W, Ling T W, Lee M L. ORA-SS: An object-relationship-attribute model for semi-structured data. 2000
16. Al-Kamha R, Embley D, Liddle S. Augmenting traditional conceptual models to accommodate XML structural constructs. In: Parent C, Schewe K-D, Storey V, Thalheim B, eds. Conceptual modeling - ER 2007. Springer Berlin Heidelberg, 2007, 518–533
17. Embley D W, Kurtz B, Woodfield S. Object-oriented systems analysis: a model-driven approach. Englewood Cliffs , New Jersey: Prentice Hall, 1992
18. Ganguly R, Sarkar A. Evaluations of conceptual models for semi-structured database system. International Journal of Computer Applications, 2012, 50(18): 5–12
19. Bird L, Goodchild A, Halpin T. Object role modelling and XML-schema. In: Laender A F, Liddle S, Storey V, eds. Conceptual modeling – ER 2000: Springer Berlin Heidelberg, 2000, 309–322
20. Curland M, Halpin T. The NORMA software tool for ORM 2. In: Soffer P, Proper E, eds. Information systems evolution. Springer Berlin Heidelberg, 2011, 190–204
21. Gao S, Sperberg-McQueen C M, Thompson H S. W3C XML schema definition language (XSD) 1.1 Part 1: structures. 2012
22. Thompson H S, Mendelsohn N, Beech D, Maloney M. XML schema Part 1: structures. 2nd ed. 2004

Amir Jahangard-Rafsanjani is currently working on his PhD in Software Engineering at Sharif University of Technology, Iran. He received an MSc in Software Engineering at Sharif University of Technology, Iran in 2008. His research is focused on model driven development, semi-structured data and formal methods.

Seyed-Hassan Mirian-Hosseinabadi received the BSc in software engineering from Shahid Beheshti University, Iran in 1984, and the MSc also in Software Engineering from Sharif University of Technology, Iran in 1987. He started his PhD program in 1993 and received the PhD in computer science (Formal Methods) from the University of Essex, UK in 1996. He joined Sharif University of Technology in 1996, and is currently an associate professor in the Department of Computer Engineering. His current research interests include application of formal methods in software specification and software development, type theory and constructive mathematics, software metrics and measurement, reconfigurable software architecture, formal specification and verification of software architecture, and XML databases.