

# Integrity Theory for Resource Space Model and Its Application<sup>1</sup>

Hai Zhuge and Yunpeng Xing

China Knowledge Grid Research Group, Key Lab of Intelligent Information Processing  
Institute of Computing Technology, Chinese Academy of Sciences, 100080, Beijing, China  
Zhuge@ict.ac.cn, Ypxing@kg.ict.ac.cn  
<http://kg.ict.ac.cn/>

**Abstract.** The Resource Space Model (RSM) is a semantic data model based on orthogonal classification semantics for effectively managing various resources in interconnection environment. In parallel with the integrity theories of relational and XML-based data models, this keynote presents the integrity theory for the RSM, including the entity integrity constraints based on the key system of the RSM, the membership integrity constraints, the referential integrity constraints, and the user-defined integrity constraints relevant to applications. This theory guarantees the RSM to correctly and efficiently specify and manage resources. Its implementation approach and application in culture exhibition are demonstrated.

## 1 Introduction

Integrity theory plays an important role in the relational data model, which has obtained a great success in both theory and systems [6, 7, 8, 10, 12, 15]. Database applications request that changes made to data by authorized users do not result in a loss of data consistency. Effective integrity constraints are means to fight the damage to data and to the consistency between data and its management mechanism. However, previous data models are limited in their abilities in effectively managing heterogeneous, distributed, and ocean resources in an open and dynamic Internet environment [17].

On the other hand, the success of World Wide Web guides people to attack the representation issue of resources. XML facilitates representation and exchange of structured information on the Internet. The Semantic Web steps further this way by using markup languages like RDF and establishing ontology mechanisms [2].

Much research on XML-based information organization and management has been done, such as Document Type Definitions, XML Schema and Unified Constraint Model for XML [3, 11, 16]. The integrity constraints for semantic specifications of XML data have drawn attention [9, 13]. However, the Internet still lacks ideal semantic data model to effectively manage distributed and expanding resources.

The Resource Space Model RSM is a semantic data model for uniformly, normally and effectively specifying and managing resources in interconnection environment.

---

<sup>1</sup> This work is supported by National Basic Research and Development Program (Semantic Grid Project, No. 2003CB317001) and the National Science Foundation of China.

Its theoretical basis is a set of normal forms on orthogonal classification semantics and the principles on relevant resource operations [17, 18, 19, 20].

Integrity is essential for the RSM to ensure its classification semantics, to maintain the consistency on resource spaces, to optimize queries, and to correctly and efficiently manage resources.

## 2 Resource Space System

The architecture of the resource space system includes five layers as depicted in Fig. 1: the application layer, the resource space system layer, the resource representation layer representing semantics of resources, the entity resource layer, and the network layer. Here focuses on the resource space system layer, which includes the resource space layer organizing resources according to semantic normal forms [17] and the operation mechanism layer implementing the operations on resources and resource spaces.

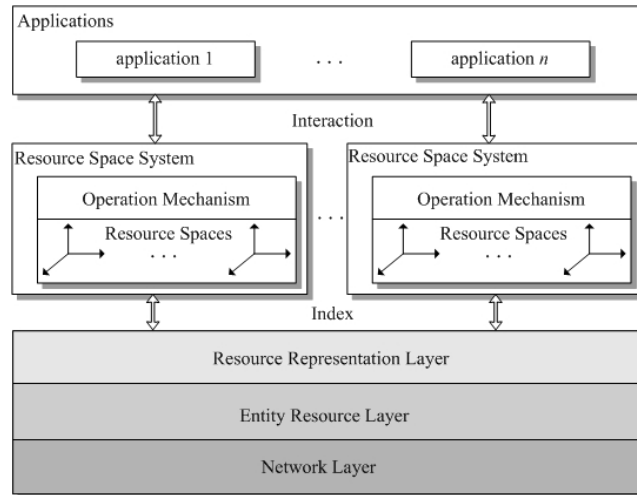


Fig. 1. The architecture of the resource space system

A resource space is an n-dimensional classification semantic space in which every point uniquely determines a set of related resources. The space is denoted as  $RS(X_1, X_2, \dots, X_n)$  or just by name  $RS$  in simple.  $X_i = \{C_{i1}, C_{i2}, \dots, C_{im}\}$  represents axis  $X_i$  with its coordinate set. A point  $p(C_{1,j1}, C_{2,j2}, \dots, C_{n,jn})$  is determined by a set of coordinate values at all axes. A point can uniquely determine a resource set, where each element is called a *resource entry*. A point can be regarded as the container of a set of resource entries.

Fig. 2 is an example of a two-dimensional resource space *Com-Goods* that specifies goods' information of three companies: Microsoft, Coca Cola and Nike. Two axes in *Com-Goods(Companies, Goods)* are *Companies* = {Microsoft, Coca Cola, Nike} and *Goods* = {soft drink, software, dress}. Each point specifies a class of goods. For example, the point (*Microsoft, software*) represents the goods belonging to

software and produced by *Microsoft*. A resource entry represents a piece of goods belonging to a point in the *Com-Goods*. Point and resource entry are two types of operation units in RSM.

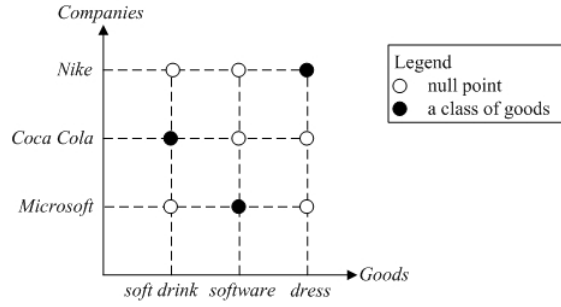


Fig. 2. An example of a two-dimensional resource space

$R(p)$  represents the resources the point  $p$  can contain. If  $R(p) = \emptyset$ , then  $p$  is called *null point*, otherwise *non-null point* (the hollow circle and solid circle in Fig. 2). Large number of null points may exist in multidimensional resource spaces. For example, the point (*Microsoft*, *soft drink*) is one of the six null points in the resource space *Com-Goods*. Here focuses on how to identify non-null points.

**DEFINITION 1.** Let  $p.X_i$  be the coordinate of  $p$  at axis  $X_i$  in  $RS(X_1, X_2, \dots, X_n)$ , i.e., the projection of  $p$  on  $X_i$ . If  $p_1.X_i = p_2.X_i$  for  $1 \leq i \leq n$  holds, then we say that  $p_1$  is equal to  $p_2$ , denoted as  $p_1 =_p p_2$ .

### 3 Entity Integrity Constraints

#### 3.1 Key

Keys play fundamental role in the relational data model and its conceptual design. They enable tuples to refer to one another and guarantee operations to accurately locate tuples [1, 4, 14].

As a coordinate system, the RSM naturally supports accurate resource location by giving coordinates. However, it is sometimes unnecessary and even arduous to specify all the coordinates to identify a non-null point, especially for high-dimensional resource spaces. The keys of RSM can further improve the efficiency of resource location.

**DEFINITION 2.** Let  $CK$  be a subset of  $\{X_1, X_2, \dots, X_n\}$ ,  $p_1$  and  $p_2$  be two non-null points in  $RS(X_1, X_2, \dots, X_n)$ .  $CK$  is called a *candidate key* of  $RS$  if we can derive  $p_1 =_p p_2$  from  $p_1.X_i = p_2.X_i$ ,  $X_i \in CK$ .

Candidate keys provide us with necessary and enough information to identify non-null points of a given resource space. Take Fig. 2 for example,  $\{Companies\}$ ,  $\{Goods\}$  and  $\{Companies, Goods\}$  are candidate keys of the resource space *Com-Goods*. Knowing the coordinates either on  $\{Companies\}$  or on  $\{Goods\}$ , we can eas-

ily get the non-null points as needed. The *primary key* is one of the candidate keys chosen by resource space designers. Any axis belonging to the primary key is called *primary axis*.

**Rule 1** (Point constraint): If axis  $X$  is one of the primary axes of the resource space  $RS$ , then for any point in  $RS$ , its coordinate on axis  $X$  should not be null.

Rule 1 is used to guarantee the primary keys' functionality of distinguishing non-null points in a given resource space. One type of null value is "at present unknown". Let  $\{Companies\}$  be the primary key of *Com-Goods*, the axis *Companies* is a primary axis of *Com-Goods*. If the coordinates of points at axis *Companies* are null, then the points (null, *soft drink*) and (null, *software*) cannot be distinguished by the primary key  $\{Companies\}$ .

In RSM, some keys can be inferred from the presence of others. This is important in query optimization, especially when dynamically creating resource spaces. Inference rules for candidate keys include the following four theorems.

**THEOREM 1.** If a set of axes  $CK$  is a candidate key of the resource space  $RS$ , then any axis set that includes  $CK$  is also the candidate key of  $RS$ .

Basic operations on resource spaces have been proposed in [18]. Here we introduce relevant definitions (definition 3-6) to develop the theory of integrity constraint.

**DEFINITION 3** (Merge of axes). If two axes  $X_1 = \{C_{11}, C_{12}, \dots, C_{1n}\}$  and  $X_2 = \{C_{21}, C_{22}, \dots, C_{2m}\}$  have the same axis name but have different coordinates, then they can be *merged* into one:  $X = X_1 \cup X_2 = \{C_{11}, C_{12}, \dots, C_{1n}, C_{21}, C_{22}, \dots, C_{2m}\}$ .

**DEFINITION 4** (Join operation). Let  $|RS|$  be the number of dimensions of  $RS$ . If two resource spaces  $RS_1$  and  $RS_2$  store the same type of resources and they have  $n$  ( $n \geq 1$ ) common axes, then they can be *joined* together as one  $RS$  such that  $RS_1$  and  $RS_2$  share these  $n$  common axes and  $|RS| = |RS_1| + |RS_2| - n$ .  $RS$  is called the join of  $RS_1$  and  $RS_2$ , denoted as  $RS_1 \cdot RS_2 \Rightarrow RS$ .

**DEFINITION 5** (Merge operation). If two resource spaces  $RS_1$  and  $RS_2$  store the same type of resources and satisfy: (1)  $|RS_1| = |RS_2| = n$ ; and (2) they have  $n - 1$  common axes, and there exist two different axes  $X_1$  and  $X_2$  satisfying the merge condition, then they can be *merged* into one  $RS$  by retaining the  $n - 1$  common axes and adding a new axis  $X_1 \cup X_2$ .  $RS$  is called the merge of  $RS_1$  and  $RS_2$ , denoted as  $RS_1 \cup RS_2 \Rightarrow RS$ , and  $|RS| = n$ .

**DEFINITION 6** (Split operation). A resource space  $RS$  can be *split* into two resource spaces  $RS_1$  and  $RS_2$  that store the same type of resources as that of  $RS$  and have  $|RS| - 1$  common axes by splitting an axis  $X$  into two:  $X'$  and  $X''$ , such that  $X = X' \cup X''$ . This split operation is denoted as  $RS \Rightarrow RS_1 \cup RS_2$ .

According to the definition of the join operation, we have the following theorem.

**THEOREM 2.** Let  $RS_1$  and  $RS_2$  be two resource spaces that can be joined together to generate a new resource space  $RS$ . Assume that  $CK_1$  and  $CK_2$  are the candidate keys of  $RS_1$  and  $RS_2$  respectively. Then,  $CK = CK_1 \cup CK_2$  is a candidate key of  $RS$ .

**Proof.** Let  $A$  be the set of all axes of  $RS$  and  $A_1$  be the set of all axes of  $RS_1$ . We assume that  $CK = CK_1 \cup CK_2$  is not the candidate key of  $RS$ . Thus, there must exist two

non-null points  $p_1$  and  $p_2$  in  $RS$ , which satisfy both  $(\forall X \in CK) (p_1.X = p_2.X)$  and  $(\exists X^* \in A) (p_1.X^* \neq p_2.X^*)$ . Without losing generality, we suppose that the axis  $X^*$  exists in  $RS_1$ . It is obvious that  $X^* \notin CK_1$ . Let  $p_1'$  be a point in  $RS_1$  which satisfies  $(\forall X \in A_1) (p_1'.X = p_1.X)$  and  $p_2'$  be a point in  $RS_1$  which satisfies  $(\forall X \in A_1) (p_2'.X = p_2.X)$ . According to the definition of *join*, we have  $R(p_1) \subseteq R(p_1')$  and  $R(p_2) \subseteq R(p_2')$  hold. So, both  $p_1'$  and  $p_2'$  are non-null points of  $RS_1$ . We can conclude that  $(\forall X \in CK_1) (p_1'.X = p_2'.X)$  and  $p_1'.X^* \neq p_2'.X^*$ . This conclusion obviously contradicts the above assumption that  $CK_1$  is the candidate key of  $RS_1$ . So  $CK = CK_1 \cup CK_2$  is a candidate key of  $RS$ .  $\square$

The following theorem can be drawn from the definition of the merge operation.

**THEOREM 3.** Let  $RS_1$  and  $RS_2$  be two resource spaces that can be merged into one resource space  $RS$ . Let  $X_1$  and  $X_2$  be two different axes of  $RS_1$  and  $RS_2$  respectively, and let  $X_c = X_1 \cup X_2$ . Assume that  $CK_1$  and  $CK_2$  are the candidate keys of  $RS_1$  and  $RS_2$  respectively. Then,  $CK = (CK_1 - \{X_1\}) \cup (CK_2 - \{X_2\}) \cup \{X_c\}$  is a candidate key of  $RS$ .

**Proof.** Let  $A$  be the set of all axes of  $RS$ . We assume that  $CK = (CK_1 - \{X_1\}) \cup (CK_2 - X_2) \cup \{X_c\}$  is not the candidate key of  $RS$ . Thus, there must exist two non-null points  $p_1$  and  $p_2$  in  $RS$ , which satisfy both  $(\forall X \in CK) (p_1.X = p_2.X)$  and  $(\exists X^* \in A) (p_1.X^* \neq p_2.X^*)$ . It is obvious that  $X^* \notin CK$ . Let  $p_1'$  and  $p_2'$  be two points in  $RS_1$  or  $RS_2$ , which satisfy  $(\forall X \in A - \{X_c\}) (p_1'.X = p_1.X)$ ,  $(\forall X \in A - \{X_c\}) (p_2'.X = p_2.X)$ ,  $p_1'.X_1 = p_1.X_c$  (or  $p_1'.X_2 = p_1.X_c$ ), and  $p_2'.X_1 = p_2.X_c$  (or  $p_2'.X_2 = p_2.X_c$ ).

Suppose that  $p_1'$  and  $p_2'$  belong to the same resource space, for example  $RS_1$ . Since  $(\forall X \in A - \{X_c\}) (p_1'.X = p_1.X)$ ,  $(\forall X \in A - \{X_c\}) (p_2'.X = p_2.X)$ ,  $p_1'.X_1 = p_1.X_c$ , and  $p_2'.X_1 = p_2.X_c$  hold, we can reach  $(\forall X \in CK_1) (p_1'.X = p_2'.X)$  and  $p_1'.X^* \neq p_2'.X^*$ . This conclusion obviously contradicts the above assumption that  $CK_1$  is the candidate key of  $RS_1$ .

Suppose  $p_1' \in RS_1$  and  $p_2' \in RS_2$ , and let  $p_2''$  be the point in  $RS_1$  that has the same coordinate values as  $p_2'$ . Thus,  $p_1'$  and  $p_2''$  have the same coordinate values as  $p_1$  and  $p_2$  respectively. We can reach that  $(\forall X \in CK_1) (p_1'.X = p_2''.X)$  and  $p_1'.X^* \neq p_2''.X^*$ . This obviously contradicts above assumption that  $CK_1$  is the candidate key of  $RS_1$ .

According to (1) and (2),  $CK = (CK_1 - \{X_1\}) \cup (CK_2 - X_2) \cup \{X_c\}$  is a candidate key of  $RS$ .  $\square$

According to the definition of the split operation, we have the following theorem.

**THEOREM 4.** Let  $RS_1$  and  $RS_2$  be two resource spaces created by splitting the resource space  $RS$ . Suppose that the axis  $X_c$  of  $RS$  is split into  $X_1$  and  $X_2$  belonging to  $RS_1$  and  $RS_2$  respectively. Let  $CK$  be a candidate key of  $RS$ . If  $X_c \notin CK$  holds, let  $CK_1 = CK_2 = CK$ , otherwise let  $CK_1 = CK - \{X_c\} \cup \{X_1\}$  and  $CK_2 = CK - \{X_c\} \cup \{X_2\}$ . Then,  $CK_1$  and  $CK_2$  are the candidate keys of  $RS_1$  and  $RS_2$  respectively.

**Proof.** Let  $A$  be the set of all axes of  $RS$  and  $A_1$  be the set of all axes of  $RS_1$ . We assume that  $CK_1$  is not the candidate key of  $RS_1$ . Thus, there must exist two non-null points  $p_1$  and  $p_2$  in  $RS_1$ , which satisfy both  $(\forall X \in CK_1) (p_1.X = p_2.X)$  and  $(\exists X^* \in A_1) (p_1.X^* \neq p_2.X^*)$ . Let  $p_1'$  and  $p_2'$  in  $RS$  have the same coordinate values as  $p_1$  and  $p_2$  respectively. It is obvious that  $(\forall X \in CK) (p_1'.X = p_2'.X)$  holds in case  $CK_1 = CK$  or  $CK_1 = CK - \{X_c\} \cup \{X_1\}$  holds.

When  $CK_1 = CK$ , if  $X^* \neq X_1$ , then  $p_1'.X^* \neq p_2'.X^*$  holds, otherwise  $p_1'.X_c \neq p_2'.X_c$  holds;

When  $CK_1 = CK - \{X_c\} \cup \{X_1\}$ , then  $X^* \neq X_1$  holds. So  $p_1'.X^* \neq p_2'.X^*$  holds.

According to (1) and (2),  $p_1' \neq_p p_2'$  does not hold. This conclusion obviously contradicts the above assumption that  $CK$  is the candidate key of  $RS$ . So  $CK_1$  is a candidate key of  $RS_1$ . Similarly, we can prove that  $CK_2$  is a candidate key of  $RS_2$ .  $\square$

In resource space systems, there often exist some resource spaces created dynamically by join, merge and split operations. Theorem 2, 3 and 4 in fact provide efficient means of deriving the candidate keys of these resource spaces created dynamically.

### 3.2 Resource Entry

In RSM, a resource entry denoted as a 3-tuple *Resource-Entry* $\langle ID, Index, Semantic-Description \rangle$  is used to index a piece of resource of resource representation layer. The first field *ID* is used to specify the resource entries in a given point. Two resource entries residing in different points could have the same ID. The second field *Index* is the index information linked to the representation layer. To facilitate semantic operations, the *Semantic-Description* uses a set of resource attributes to reflect the simple semantics of the resource in the given resource space. The resource representation layer describes the detailed semantics of all resources. In the following discussion,  $re.ID$ ,  $re.index$  and  $re.SD$  denote the *ID*, *Index* and *Semantic-Description* of resource entry  $re$  respectively.

Three types of entity integrity constraints for resource entries are presented as follows. All resource space systems are required to satisfy the first two rules, and the third one is optional according to application requirement.

**Rule 2** (Resource entry constraint 1): Any ID should not be null, and for any two resource entries  $re_1$  and  $re_2$  in the same non-null point,  $re_1.ID \neq re_2.ID$  holds.

Rule 2 requires that all resource entries in a given non-null point should be distinguishable through IDs. This is helpful to guarantee that any operation can precisely locate the target resource entry.

**Rule 3** (Resource entry constraint 2). Any index of a resource entry should not be null, and for any two resource entries in the same non-null point  $re_1$  and  $re_2$ ,  $re_1.index \neq re_2.index$ .

Rule 3 requires the following two conditions: (1) any resource entry should include the index information linked to the representation layer, and (2) there should not exist two different resource entries that have identical index information in a given non-null point. Otherwise, it will lead to information redundancy and unnecessary maintenance of consistency between resource entries in a given point.

The syntactic structure of index information of resource entries depends on the implementation of resource representation layer. For instance, the XML-based implementation of resource representation layer usually uses XPath expressions [5], whereas filenames are often employed for the file-based implementation. To analyze the indexes of resource entries, not only the syntactic structure but also the semantics should be considered. For example, the absolute path differs from the relative path syntactically. However, these two types of paths may represent the same index information.

**Rule 4** (Resource entry constraint 3): Any semantic description  $SD$  of resource entry should not be null, and for any two resource entries  $re_1$  and  $re_2$  in the same non-null point, they are not the same and do not imply each other in semantics (i.e., neither  $re_1.SD \Rightarrow re_2.SD$  nor  $re_2.SD \Rightarrow re_1.SD$ ).

Rule 4 is the entity integrity constraint about the *Semantic-Description* of a resource entry. It is optional and stricter than Rule 3. Since  $re.SD$  determines the semantic existence of the resource entries in the resource space, Rule 4 requires that  $re.SD$  should not be null. Furthermore, resource entries in a given non-null point should not be the same or imply each other in semantics. For example, a resource and its copies are allowed to coexist in a given non-null point by Rule 3, but not by Rule 4.

## 4 Membership Integrity Constraint

In relational databases, a tuple can be inserted into a table only if all fields of the tuple satisfy the corresponding domain constraints. So the membership between the tuple and the table should be judged before operation. In RSM, a resource space represents the classification semantics of resources. The existence of resource entry  $re$  in point  $p$  means that the resource indexed by  $re$  belongs to the type represented by  $p$ . A resource entry can be inserted into a point by the following operation statement:

**INSERT**  $re\langle ID, index, Semantic-Description \rangle$  **INTO**  $p(C_{1,i1}, C_{2,i2}, \dots, C_{n,in})$ .

Without any restriction, even a resource entry representing a car could be inserted into the point (*Microsoft, Software*) of the resource space shown in Fig. 2. So, checking the memberships of resource entries plays an important role in RSM.

Note that  $R_\Delta(\mathbf{RS})$ ,  $R_\Delta(C)$  and  $R_\Delta(p)$  denote the sets of resources currently stored by resource space  $\mathbf{RS}$ , coordinate  $C$  and point  $p$  respectively. For any resource entry  $re$ , if  $re$  has been inserted into the point  $p$ , then  $re \in R_\Delta(p)$ .

**Rule 5** (Membership constraint): Let  $re\langle ID, index, Semantic-Description \rangle$  be a resource entry. For any point  $p(C_{1,i1}, C_{2,i2}, \dots, C_{n,m})$  in a given resource space,  $re \in R_\Delta(p) \rightarrow re \in R(p)$  holds.

As illustrated in Rule 5, a resource entry  $re$  can be inserted into point  $p$  only if  $re$  really belongs to the type that  $p$  represents. This membership integrity constraint can avoid incorrect resource classification. When the insert operation or update operation on resource entries is involved, this constraint should be verified.

## 5 Referential Integrity Constraints

Relational database applications often require that a value appeared in one relation for given set of attributes should also appear for a certain set of attributes in another relation. This condition is called referential integrity constraint. In the following, three types of referential integrity constraints for the RSM are given.

### 5.1 Referential Integrity for Resource Entries

In RSM, the basic function of a resource entry is to index a certain resource in resource representation layer. For any resource entry  $re\langle ID, index, Semantic-$

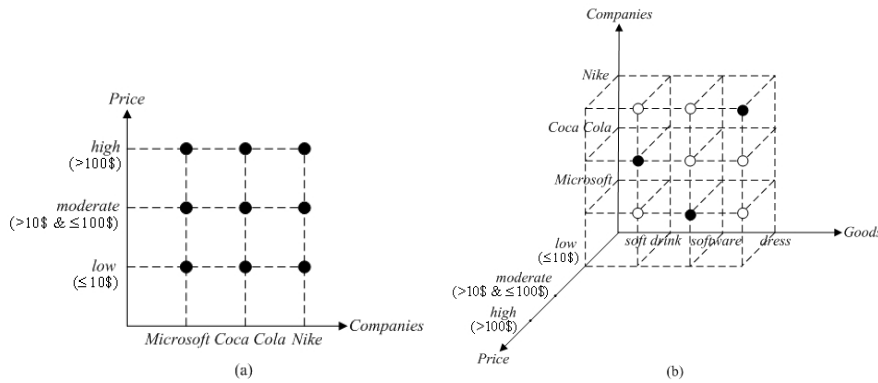
*Description*, *re.index* represents the index information of the corresponding resource. In section 3.2, Rule 3 ensures that *re.index* is non-null. But it cannot guarantee that *re.index* makes sense. This is mainly because some modifications on resource entries or resource representation layer may cause indexes of resource entries to be dead links. The following referential integrity constraint is to eliminate dead links.

**Rule 6** (Referential constraint 1). For any resource entry *re* in a given resource space system, there exists a resource in the resource representation layer which is referred by the index information (*re.index*).

The resource space layer is the referencing layer and the resource representation layer is the referenced layer. Rule 6 guarantees that *re.index* makes sense for any resource entry *re*. This constraint should be checked when the insertion of *re* or the update of *re.index*. When delete operation takes place in the resource representation layer, this integrity also needs to be checked.

### 5.2 Referential Integrity Between Resource Spaces

The first type of referential integrity constraint between resource spaces is relevant to the join operation. In Fig. 3(a), the two-dimensional resource space *Price-Com* stores the same type of resources as the resource space *Com-Goods* (see Fig. 2.). The two resource spaces have a common axis *Companies*, so they can be joined together. Fig. 3(b) depicts the resource space *Com-Goods-Price* created by joining *Price-Com* and *Com-Goods*.



**Fig. 3.** A two-dimensional resource space and a three-dimensional resource space

All resource entries of *Com-Goods-Price* come from *Price-Com* and *Com-Goods*. *Com-Goods-Price* provides further classification for these resource entries. With no doubt, there exists certain referential relation among *Com-Goods-Price*, *Price-Com* and *Com-Goods*.

**Rule 7** (Referential constraint 2). Let  $RS_1$ ,  $RS_2$  and  $RS$  be three resource spaces that satisfy  $RS_1 \cdot RS_2 \Rightarrow RS$ , then  $R_\Delta(RS) \subseteq R_\Delta(RS_1) \cup R_\Delta(RS_2)$  must hold.

$RS$  is derived from  $RS_1$  and  $RS_2$ , and Rule 7 maintains the dependency of  $RS$  on  $RS_1$  and  $RS_2$ . Thus, when a resource entry is inserted into  $RS$  or deleted from  $RS_1$  or  $RS_2$ , this constraint should be checked.

The second type of referential integrity constraint between resource spaces assumes that the involved resource spaces satisfy the third-normal-form. We first define the foreign key for RSM.

**DEFINITION 7.** Let  $S$  be a subset of axes of the resource space  $RS_1$ , and it is not the primary key of  $RS_1$ . If there exists another resource space  $RS_2$  such that  $R(RS_1) \subseteq R(RS_2)$  holds and  $S$  is the primary key of  $RS_2$ , then  $S$  is called the *foreign key* of the resource space  $RS_1$ .  $RS_1$  is called the referencing resource space of  $RS_2$  and  $RS_2$  is called the referenced resource space of  $RS_1$ .

According to definition 7, we have the following theorem.

**THEOREM 5.** Let  $S = \{X_1, X_2, \dots, X_m\}$  be the foreign key of the referencing resource space  $RS_1(X_1, X_2, \dots, X_m, X_{m+1}, \dots, X_n)$ , and  $RS_2(X_1, X_2, \dots, X_m, Y_{m+1}, \dots, Y_l)$  be the corresponding referenced resource space. For two non-null points  $p(C_1, C_2, \dots, C_m, C_{m+1}, \dots, C_n)$  and  $p'(C_1, C_2, \dots, C_m, C'_{m+1}, \dots, C'_l)$  in  $RS_1$  and  $RS_2$  respectively,  $R(p) \subseteq R(p')$  holds.

**Proof.** It is obvious that  $R(p') = R(C_1) \cap R(C_2) \cap \dots \cap R(C_m) \cap R(C'_{m+1}) \cap \dots \cap R(C'_l)$ . Because  $RS_2$  satisfies the third-normal-form [18, 19] and  $S$  is the primary key of  $RS_2$ ,  $R(p') = R(C_1) \cap R(C_2) \cap \dots \cap R(C_m)$  holds. Since  $R(p) = R(C_1) \cap R(C_2) \cap \dots \cap R(C_m) \cap R(C_{m+1}) \cap \dots \cap R(C_n)$  holds, we have  $R(p) \subseteq R(C_1) \cap R(C_2) \cap \dots \cap R(C_m)$ . So  $R(p) \subseteq R(p')$  holds.

The theorem 5 indicates the inclusion relation between the points in the referencing resource space and their counterparts in the referenced resource space. The following constraint is to maintain the legal referential relation between the referencing resource space and its referenced resource space.

**Rule 8** (Referential constraint 3). Let  $S = \{X_1, X_2, \dots, X_m\}$  be the foreign key of the referencing resource space  $RS_1(X_1, X_2, \dots, X_m, X_{m+1}, \dots, X_n)$ , and  $RS_2(X_1, X_2, \dots, X_m, Y_{m+1}, \dots, Y_l)$  be the corresponding referenced resource space. For two non-null points  $p(C_1, C_2, \dots, C_m, C_{m+1}, \dots, C_n)$  and  $p'(C_1, C_2, \dots, C_m, C'_{m+1}, \dots, C'_l)$  in  $RS_1$  and  $RS_2$  respectively,  $R_\Delta(p) \subseteq R_\Delta(p')$  holds.

Rule 8 guarantees that if a resource entry  $re$  appears in a point  $p$  in the referencing resource space, then  $re$  must exist in the counterpart of  $p$  in the referenced resource space.

## 6 User-Defined Integrity Constraints

Any resource space systems should conform to entity integrity constraints, membership integrity constraints and referential integrity constraints. In specific applications, different resource space systems should obey different context-relevant constraints. These constraints are called user-defined integrity constraints. Here introduces three frequently used types of user-defined constraints. Two resource spaces shown in Fig. 4 are used to illustrate the user-defined and application-relevant integrity constraints. In Fig. 4(a), the resource space *Age-Gender* is used to accommodate employees' information. Every point classifies these employees by their age and gender. In Fig. 4(b), resource space *Tutor-Class* is used to represent students' information. Each point of *Tutor-Class* classifies these students by their tutor and class information.

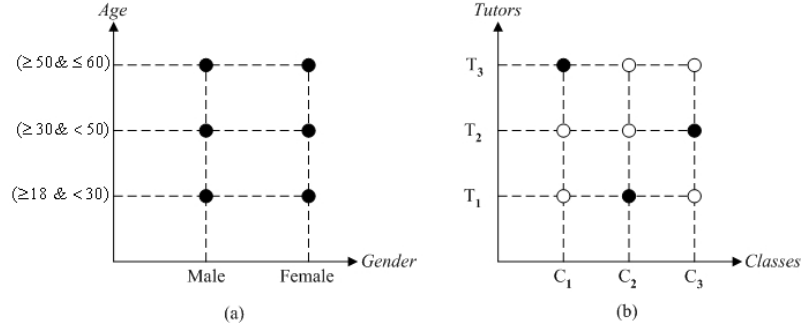


Fig. 4. Examples of two-dimensional resource space

### 6.1 Value-Based Constraint

This type of user-defined constraints requires the attributes' values in resource description to satisfy some rules. Function  $GetValue(attr)$  returns the value of attribute  $attr$  specified in the *Semantic-Description* of a certain resource entry. This type of constraints can be described as follows:

$$\begin{aligned}
 \langle \text{Constraint expression} \rangle & ::= \langle \text{Operand} \rangle \langle \text{Relation-Op} \rangle \langle \text{Operand} \rangle \mid \\
 \langle \text{Constraint expression} \rangle & \quad \vee \quad \langle \text{Constraint expression} \rangle \mid \\
 \langle \text{Constraint expression} \rangle & \quad \wedge \quad \langle \text{Constraint expression} \rangle \mid \\
 \neg \langle \text{Constraint expression} \rangle; \\
 \langle \text{Operand} \rangle & ::= GetValue(attr) \mid \text{user-defined-constant-value}; \\
 \langle \text{Relation-Op} \rangle & ::= < \mid > \mid = \mid \leq \mid \geq \mid \neq.
 \end{aligned}$$

Take Fig. 4(a) for example, if the designer requires that any male employee should not be older than 70 and female should not be older than 60 in *Age-Gender*, then this user-defined constraint can be described as follows:

For any resource entry  $re$ ,

$$\begin{aligned}
 (GetAttribute(gender) = \text{"male"} \wedge GetAttribute(age) \leq 60) \vee \\
 (GetAttribute(gender) = \text{"female"} \wedge GetAttribute(age) \leq 55) \text{ holds.}
 \end{aligned}$$

Before the resource entry  $re$  can be inserted into *Age-Gender* or updated, the system should check whether the above constraint has been violated.

### 6.2 Resource-Entry-Based Constraint

In some applications, semantic relations among resource entries should be considered. Operations on a resource entry may require other operations on semantically relevant resource entries. This type of user-defined constraints is called resource-entry-based constraint. For example,  $RS$  is a resource space that contains all registration information of students in a school and  $RS'$  is a resource space that contains all health information of these students. Let  $re$  be the resource entry representing a student's registration information and  $re'$  be the resource entry representing his/her health information. The health information depends on the valid registration information, i.e.,  $re' \in R_{\Delta}(RS') \rightarrow re \in R_{\Delta}(RS)$  must hold. So this constraint should be checked before the insertion of  $re'$  or after the deletion of  $re$ .

### 6.3 Point-Based Constraints

As resource sets, points are often required to satisfy some application relevant rules from the viewpoint of set theory. Take Fig. 4(b) for example, suppose that a class has only one tutor in *Tutor-Class* and that each tutor is in charge of only one class. For any  $T_i$ , there at most exists one  $C_j$  such that  $R_\Delta(\mathbf{p}(T_i, C_j)) \neq \emptyset$ , and for any  $C_m$  there at most exists one  $T_n$  such that  $R_\Delta(\mathbf{p}(T_n, C_m)) \neq \emptyset$ . We define function

$$NotNull(\mathbf{p}) = \begin{cases} 1, & R_\Delta(\mathbf{p}) \neq \emptyset \\ 0, & R_\Delta(\mathbf{p}) = \emptyset \end{cases} \text{ and use } \mathbf{p}_{ij} \text{ to denote the point } \mathbf{p}(T_i, C_j). \text{ Then, this constraint can be formally represented as: } \forall i (\sum_{j=1}^3 NotNull(\mathbf{p}_{ij}) \leq 1) \wedge$$

$\forall j (\sum_{i=1}^3 NotNull(\mathbf{p}_{ij}) \leq 1)$ . Thus, before any student information can be inserted into

*Tutor-Class*, the system needs to check whether the above constraint is violated or not.

## 7 Implementation

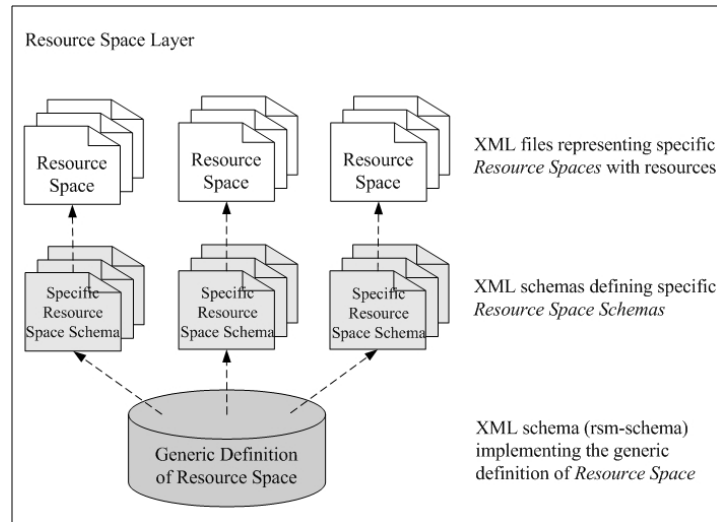
A resource space system includes functions for resource space definition, manipulation and system management. Its underlying metadata and data structure use XML and XML schema as shown in Fig. 5. An xml schema *rsm-schema* given at [kg.ict.ac.cn/rsm/rsm-schema.xsd](http://kg.ict.ac.cn/rsm/rsm-schema.xsd) specifies the generic definitions of resource space such as *resource space*, *axis*, *coordinate*, and *integrity constraint*. The domain resource space schema is the derivation of the *rsm-schema*. Resource spaces can be specified by the XML files, which are regulated by domain specific schemas. The following statement defines resource spaces:

```

create resource_space rsname (
  (axis_name(coord_name[, coord_name]...)
  [, axis_name(coord_name[, coord_name]...) ]...)
  primary key (axis_name[, axis_name]...)
  [no_sem_duplication]
  [[foreign key (axis_name[, axis_name]...) references rsname]...]
  [join_ref (rsname[, rsname]...)]
  [[check expression]...]
)

```

The users and applications can define the schemas of resource spaces and specify which optional constraints these resource spaces should obey besides the required ones. The clauses “*no\_sem\_duplication*”, “*join\_ref (...)*”, “*foreign key (...)* *reference...*”, and “*check...*” represent that the target resource space should comply with the entity constraint on entry semantic description, the Join operation relevant reference constraints, the foreign key based reference constraints and the user-defined



**Fig. 5.** The implementation of the Resource Space

constraints respectively. Here demonstrates how the integrity constraints of RSM are expressed in the *rsm-schema* of the resource space system.

**Point Constraint.** The following XML definition defines the axes of RSM. The statement “`<xs:attribute name="iskey" type="xs:boolean"/>`” defines a flag for each axis from which the resource space system can judge whether a certain axis is one of the primary axes of a resource space. Therefore, the resource space system can determine the primary key of a given resource space.

```

<xs:complexType name="rsm:Axis">
  <xs:sequence>
    <xs:element ref="rsm:coorHierarchy" maxOccurs="unbounded"/>
    <!-- rsm:coorHierarchy represents the definition of coordinate hierarchy -->
  </xs:sequence>
  <xs:attribute name="axname" type="xs:string" use="required"/>
  <xs:attribute name="iskey" type="xs:boolean"/>
</xs:complexType>

```

**Resource Entry Constraint.** The first part of the following definition is the entry in *rsm-schema*: `<ID, Index, Semantic-Description>`. In the second part, each point of a resource space consists of a number of resource entries. The code in bold has defined two types of “`xs:key`” in “*point*”. Therefore, the resource space system makes use of the feature of “`xs:key`” in XML schema to guarantee the uniqueness of *ID* and *Index* of resource entries respectively. Since resource entry constraint 3 is optional, the third part has defined a boolean variable “*IC-sem-entry*” through which the resource space system can determine whether a resource space need to keep this semantic description constraint.

```

<xs:complexType name="Entry">
  <xs:sequence>
    <xs:element ref="rsm:entryID"/>
    <xs:element ref="rsm:entryIndex"/>
    <xs:element ref="rsm:entrySD"/>
  </xs:sequence>
</xs:complexType>
.....
<xs:complexType name="Point">
  <xs:sequence>
    <xs:element ref="rsm:axes"/>
    <xs:element ref="rsm:entry" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="point" type="rsm:Point">
  <xs:key name="KeyOfEntryID">
    <xs:selector xpath="rsm:entry"/>
    <xs:field xpath="rsm:entryID"/>
  </xs:key>
  <xs:key name="KeyOfEntryIndex">
    <xs:selector xpath="rsm:entry"/>
    <xs:field xpath="rsm:entryIndex"/>
  </xs:key>
</xs:element>
.....
<xs:simpleType name="IC-sem-entry">
  <xs:restriction base="xs:boolean"/>
</xs:simpleType>

```

**Membership Constraint** guarantees a resource entry to fall into the proper point in a resource space. Although there is no corresponding code to reflect the membership constraint in the *rsm-schema*, the resource space system can provide a mechanism to make sure that all resource spaces obey this constraint. Before a new resource entry is inserted into a point or after an existing resource entry of a point is updated, the resource space system needs to retrieve the corresponding features of the resource over the point to check whether the resource belongs to this point. If the membership constraint is violated, the insert or update operation should be cancelled.

**Reference Constraint.** The reference constraint 1 requests that the resource entity corresponding to the index of a resource entry in a resource space must be represented in the resource representation layer. Once modification to URIs of resource entities in the representation layer or to the indices of resource entries takes place, the resource space system needs to check whether the reference constraint 1 is violated.

As for the reference constraints 2 and 3, using the code in bold in the first part of the following XML schema definition, the resource space system can record the reference relationship between resource spaces. Thus, once resource entries are inserted into referring resource spaces or removed from referred resource spaces, the resource space system will check whether the referring resource spaces and the referred resource spaces are compatible with the reference relationship.

```

<xs:simpleType name="RefSpace">
  <xs:restriction base="xs:anyURI"/>
</xs:simpleType>
<xs:complexType name="IC-ref-join">
  <xs:sequence>
    <xs:element name="joiningSpace" type="rsm:RefSpace"/>
    <xs:element name="joiningSpace" type="rsm:RefSpace"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="IC-foreign">
  <xs:sequence>
    <xs:element name="foreignSpace" type="rsm:RefSpace"/>
  </xs:sequence>
</xs:complexType>
.....
<xs:complexType name="IC-userdefined">
  <xs:sequence>
    <xs:element ref="rsm:check" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

**User-defined Constraint.** In the second part of above XML schema definition, the resource space system uses “*rsm:check*” elements to record all the user-defined constraints a resource space should comply with. Once any of these constraints is broken, the resource space system should cease the operations. The following is the definition of “*Constraint*” element in *rsm-schema*.

```

<xs:complexType name="Constraint">
  <xs:sequence>
    <xs:element name="ic-sem-entry" type="rsm:IC-sem-entry"/>
    <xs:element name="ic-ref-join" type="rsm:IC-ref-join" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="ic-foreign" type="rsm:IC-foreign" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="ic-userdefined" type="rsm:IC-userdefined" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

```

Where the element “*Constraint*” is defined to totally specify which optional constraints should be complied with besides the required ones. The optional constraints include entity constraint about entry semantic description (“*ic-sem-entry*”), Join operation related reference constraints (“*ic-ref-join*”), foreign key based reference constraints (“*ic-foreign*”), and user-defined constraints (“*ic-userdefined*”).

## 8 Application in Dunhuang Culture Exhibition

Dunhuang of west China includes over 1000 ancient caves containing precious wall-painting and color statues. Our project Dunhuang Culture Grid is to exhibit the artifacts on the Internet by using the technologies of animation, virtual reality and the

Knowledge Grid [19]. The fundamental work is to establish the resource spaces for the artifacts in caves and to set their integrity constraints. Fig. 6 illustrates the resource space modeling of a cave in contrast to the relational modeling. The resource space is based on classification semantics. A point in a well-defined resource space uniquely determines a class of artifacts.

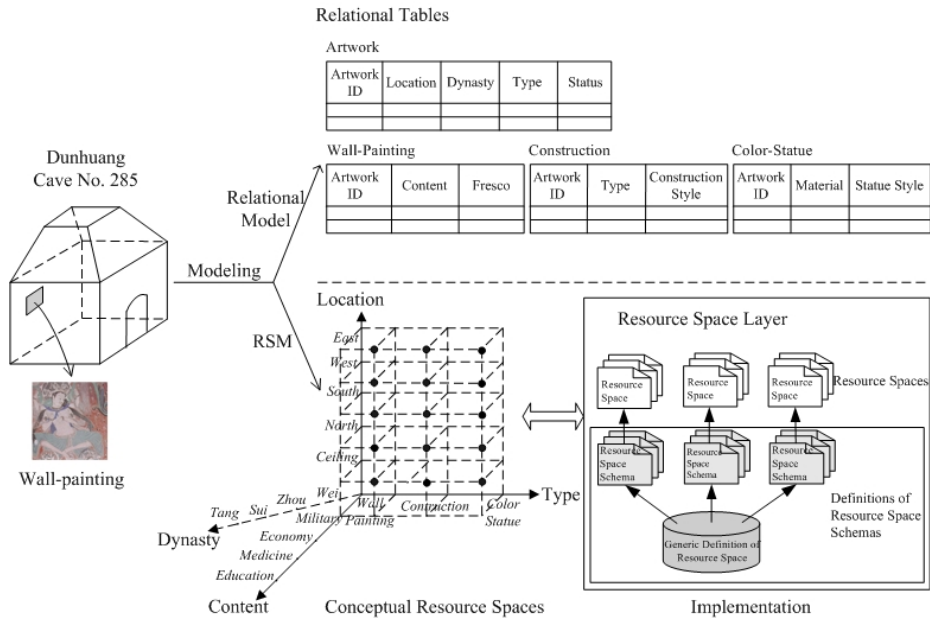


Fig. 6. RSM modeling in contrast to relational modeling

The Dunhuang culture resource space schemas derived from the *rsm-schema* are given at [kg.ict.ac.cn/rsm/dh/rsm-dunhuang.xsd](http://kg.ict.ac.cn/rsm/dh/rsm-dunhuang.xsd). Here uses two resource spaces of the system to illustrate how to define the integrity constraints for resource spaces. The resource space *fresco-285* is to specify all the wall-paintings in cave no. 285. The following is the creation statement:

```

create resource_space fresco-285 (
    (dynasty(Wei, Zhou, Sui, Tang),
    location(east, south, west, north),
    status(spoilt, maintained, available)
    )
    primary key (dynasty, location, status)
    no_sem_duplication
)
    
```

All its dimensions *dynasty*, *location* and *status* constitute the primary key. This resource space also requires that the entity constraint about the entry semantic description needs to be maintained. A resource space *flying-deity-285* described below is to manage the unspoiled flying deities in cave no. 285.

```

create resource_space flying-deity-285 (
    (dynasty(Wei, Zhou, Sui, Tang),
    location(east, south, west, north),
    status(spoilt, maintained, available),
    flying-style(childish, naked, flowered)
    )
    primary key (dynasty, location, status, flying-style)
    no_sem_duplication
    foreign key (dynasty, location, status) references fresco-285
    check status≠'spoilt'
)
    
```

flying-deity-285 has four dimensions: *dynasty*, *location*, *status* and *flying-style*, they constitute the primary key. The axes *dynasty*, *location* and *status* are the foreign key that results in the reference from *flying-deity-285* to *fresco-285*. A user-defined constraint “*status*≠'spoilt'” has been defined to indicate that *flying-deity-285* can only contain the unspoiled flying deities.

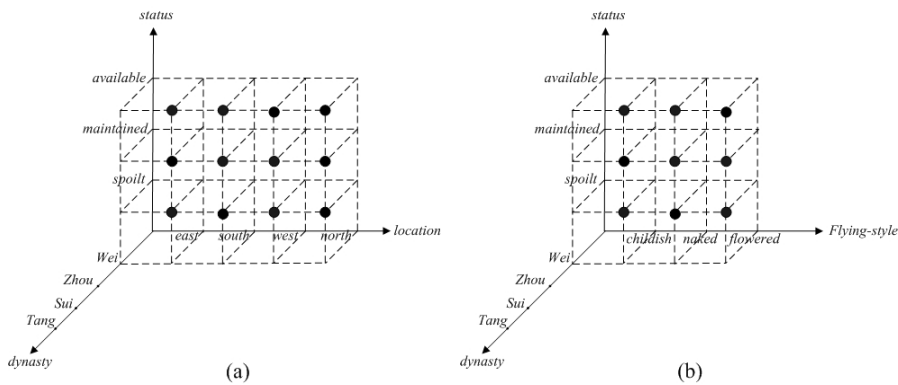


Fig. 7. The resource space fresco-285 (a) and a slice of the resource space flying-deity-285(b)

Fig. 7 (b) corresponds to the *west* coordinate in the axis *location*. The integrated XML files representing *fresco-285* and *flying-deity-285* are available at [kg.ict.ac.cn/rsm/dh/fresco-285.xml](http://kg.ict.ac.cn/rsm/dh/fresco-285.xml) and [kg.ict.ac.cn/rsm/dh/flying-deity-285.xml](http://kg.ict.ac.cn/rsm/dh/flying-deity-285.xml).

## 9 Conclusion

Based on our resource space model RSM, this keynote proposes the integrity theory for RSM. It is the basis for guaranteeing the correctness of operations on resource spaces. The implementation approach and application in culture area demonstrate its usability. More background and practice on Dunhuang Cultural Grid and Knowledge Grid are available at [www.culturegrid.net](http://www.culturegrid.net) and [www.knowledgegrid.net](http://www.knowledgegrid.net). We are investigating the integrity on a new semantic model integrating RSM with SLN (a Semantic Link Network model [19]).

## References

1. Abiteboul, S., Hull, R. and Vianu, V.: Foundations of Databases. Addison-Wesley, Reading, MA (1995).
2. Berners-Lee, T., Hendler, J. and Lassila, O.: Semantic Web. *Scientific American*, 284 (5) (2001) 34-43.
3. Bray, T., Paoli, J. and Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0. W3C Recommendation, February 1998. [www.w3.org/TR/REC-xml/](http://www.w3.org/TR/REC-xml/).
4. Buneman, P. et al.: Keys for XML. In: *Proceedings of International World Wide Web Conference, WWW10*, Hong Kong, 1-5 May 2001.
5. Clark, J. and DeRose, S.: XML Path Language (XPath). W3C Working Draft, November 1999. [www.w3c.org/TR/xpath](http://www.w3c.org/TR/xpath).
6. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13 (6) (1970) 377-387.
7. Codd, E.F.: Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4 (4) (1979) 397-434.
8. Date, C.J.: Referential Integrity. In: *Proceedings of International Conference on Very Large Data Bases*, Cannes, 9-11 September 1981.
9. Davidson, A. et al.: Schema for Object-Oriented XML 2.0. W3C Note, July 1999. [www.w3c.org/TR/NOTE-SOX](http://www.w3c.org/TR/NOTE-SOX).
10. Eswaran, K.P. and Chamberlin, D. D.: Functional Specifications of a Subsystem for Database Integrity. In: *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
11. Fan, W. Kuper, G. and Simon, J.: A Unified Constraint Model for XML. In: *Proceedings of International World Wide Web Conference, WWW10*, Hong Kong, 1-5 May 2001.
12. Hammer, M.M. and Mcleod, D.J.: Semantic Integrity in a Relational Data Base System. In: *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
13. Layman, A. et al.: XML-Data. W3C Note, January 1998. [www.w3c.org/TR/1998/NOTE-XML-data](http://www.w3c.org/TR/1998/NOTE-XML-data).
14. Ramakrishnan, R. and Gehrke, J.: Database Management Systems. McGraw-Hill Higher Education, New York, 2000.
15. Stonebraker, M.: Implementation of Integrity Constraints and Views by Query Modification. In: *Proceedings of ACM-SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
16. Thompson, H.S. et al.: XML Schema. W3C Working Draft, May 2001. [www.w3c.org/XML/Schema](http://www.w3c.org/XML/Schema).
17. Zhuge, H.: Resource Space Model, Its Design Method and Applications. *Journal of Systems and Software*, 72 (1) (2004) 71-81.
18. Zhuge, H.: Resource Space Grid: Model, Method and Platform. *Concurrency and Computation: Practice and Experience*, 16 (14) (2004) 1385-1413.
19. Zhuge, H.: The Knowledge Grid, *World Scientific*, Singapore, 2004.
20. Zhuge, H., Yao, E., Xing, Y. and Liu, J.: Extended Normal Form Theory of Resource Space Model. *Future Generation Computer Systems*, 21 (2005) 189-198.