Journal of King Saud University – Computer and Information Sciences (2015) 27, 181–192

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

Transformation rules for decomposing heterogeneous data into triples

Mrityunjay Singh *, S.K. Jain

National Institute of Technology, Kurukshetra 136119, India

Received 8 May 2013; revised 6 February 2014; accepted 13 March 2014
Available online 26 March 2015

KEYWORDS

Information integration;
Dataspace system;
Triple model;
Heterogeneity;
Transformation Rules Set;
Data modeling

Abstract In order to fulfill the vision of a dataspace system, it requires a flexible, powerful and versatile data model that is able to represent a highly heterogeneous mix of data such as databases, web pages, XML, deep web, and files. In literature, the triple model was found a suitable candidate for a dataspace system, and able to represent structured, semi-structured and unstructured data into a single model. A triple model is based on the decomposition theory, and represents variety of data into a collection of triples. In this paper, we have proposed a decomposition algorithm for expressing various heterogeneous data models into the triple model. This algorithm is based on the decomposition theory of the triple model. By applying the decomposition algorithm, we have proposed a set of transformation rules for the existing data models. The transformation rules have been categorized for structured, semi-structured, and unstructured data models. These rules are able to decompose most of the existing data models into the triple model. We have empirically verified the algorithm as well as the transformation rules on different data sets having different data models.

1. Introduction

In recent past, the attention has been made on the efficient management of the large volume of heterogeneous data distributed over several sites. Data integration is one way for managing such large collection of heterogeneous data but it has various shortcomings (Dong et al., 2009; El-Sappagh et al., 2011; Lenzerini, 2002). Recently , the dataspace approach has emerged as a new way of data integration which integrates the heterogeneous data in "pay-as-you-go" manner (Halevy et al., 2006; Franklin, 2009). This approach provides an incremental improvement over the existing data management systems for managing and querying the heterogeneous data in a uniform manner (Hedeler et al., 2009; Mirza et al., 2010). A dataspace is defined as a set of participants and a set of relationships among them. A participant may be any data source which contains data and may vary from structured to unstructured (Franklin et al., 2005; Singh and Jain, 2011). The examples of a dataspace system include Personal Information Management (PIM) (Dittrich et al., 2006; Dittrich et al., 2007), Scientific Data Management (Dessì and

* Corresponding author. Tel.: +91 8295594224.
E-mail addresses: mrityunjay.cse045@gmail.com (M. Singh), skj_nith@yahoo.com (S.K. Jain).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2014.03.017

Pes, 2009; Elsayed and Brezany, 2010), management of structured data on web such as Linked Data (Bizer et al., 2009; Ngomo, 2012; Van Hage et al., 2012).

The development of a dataspace system requires a simple and flexible data model for uniform representation of the heterogeneous data in a data-space. Previously, Halevy et al. have argued that a semi-structured graph based model is more suitable for dataspace systems (Halevy et al., 2006). Zhong et al. have advocated the use of Resource Description Framework (RDF) (Zhong et al., 2008) and proposed the *triple model* based on the RDF data model. A triple model is a simple and flexible data model based on the decomposition theory, which represents the heterogeneous data in data-space without losing their semantics. This model decomposes a large data unit into a set of smaller data units, and encapsulates each data unit into a triple.

In order to express various data models in triple model, and to avoid the uncertainty in data at various levels, a set of translation rules is required. In this work, we have employed the novel decomposition theory of triple model and proposed an algorithm which decomposes a data model into a collection of triples. Our algorithm works in two phases: *phase-1,* identifying all data item classes belonging to the input data model, and *phase-2,* decomposing each class to their respective components and encapsulating each component into a set of triples. Based on the decomposition algorithm, we have proposed a set of transformation rules for the structured, semi-structured and unstructured data models.

Previously, Zhong et al. present a set of decomposition rules w.r.t. a few data models (Zhong et al., 2008), whereas our work comprises of presenting a large set of transformation rules and a decomposition algorithm to apply on them. The proposed Transformation Rules Sets (TRSs) are exhaustive, and cover a broad range of data models in practical use. Therefore, these rules sets form a good base for implementation. One can extend these TRSs as well as the decomposition algorithm for other data models by identifying their respective classes and properties. We have applied our TRSs on various kinds of existing data models such as object relational, XML, iDM data model.

The rest of the paper is organized as follows: Section 2 presents the basic idea of the triple model. TRSs for various kinds of data models are presented in Section 3. The comparison and discussion of work are presented in Section 4 and 5 respectively. We have concluded our work in Section 6.

2. Triple model

A triple model is a graph based data model in which the smallest modeling unit is a triple. A *triple (T)* has three tuples (S, P, O), where S is a subject component, P is a predicate component, and O is an object component. Subject component(S) is a unique identifier of a data item, which is an integer type. Predicate component(P) has a 2-tuples (l, d), where l is a finite string that represents the label, and d is also a finite string which represents the data type. Object component(O) stores the actual data as an byte array.

A *data item (π)* is a unit populated in a dataspace which constitutes data such as a real world entity, a relation, a tuple, an xml element, a database, a file/folder, a web page. Before populating a data item in a dataspace, it must be decomposed into a collection of triples. For example, before populating the employee data item (e_1) in a dataspace, it must be decomposed into a set of triples as {(e_1, (*emp_name*, string), "R. Kumar"), (e_1, (*date_of_birth*, date), "17 /11/1983"), (e_1, (*date_of_joining*, date), "15/07/2009"), (e_1, (*organization*, string), "NIT"), (e_1, (*department*, string), "Computer engineering department"), and (e_1, (*salary*, currency), Rs 41,543/-)} as shown in Fig. 2.

A *data item class $C(\pi)$* is the predefined class for a data item. The set of data items having common properties are grouped into a data item class, e.g., files, folders, relations, XML elements, objects, web pages, an abstract entity like person. Every data item in a dataspace must belong to a predefined data item class otherwise we define a new class for this data item, e.g., a resource view class for a resource view data item in iDM model (Dittrich and Salles, 2006).

A *triple graph(G)* is a logical graph which is constructed among different triples populated in a dataspace. The triple graph (G) is defined as $G = (N, E, L)$, where N is a set of nodes. The internal nodes represent a data item with their identification, the leaf nodes represent the literal values which contain data. E is a set of edges. As shown in Fig. 1, an edge represents a relationship between either two data items (*i.e., association edge*) or a data item and its value (*i.e., attribute edge*) w.r.t property P. The association edge is represented as $< dataitem, association, dataitem >$, and the attribute edge is represented as $< dataitem, property, value >$. L is a set of labels on an edge with attribute or association name. Fig. 2 illustrates an example of triple graph in which the internal nodes are represented by an oval, and leaf nodes are represented by a dotted oval, a label on edge represents the predicate component of triple, and the direction of arrow is from subject to object of a triple.

A *Transformation Rule (TR)* maps a data model into the triple model without losing the semantics of data. The TRs for a data model depend on its respective properties. The collections of TRs related to a single data item class are grouped into the *Transformation Rules Set (TRS)*.

A *wrapper* is a program which extracts the desired data from its respective data sources, and transforms them into a collection of triples. A wrapper has two modules: a data extractor module and a data translator module. The data extractor module extracts the desired data from its respective data sources whereas the data translator module is based on TRSs, and translates the extracted data into a collection of triples. We have implemented a set of automatic/semi-automatic wrappers for the verification of the TRSs w.r.t. few data models such as structured data models (e.g., MySQL, PostgresSQL databases etc.), semi-structured data models (e.g., XML data, file system data, bibliographic data, LATEX data etc.), and unstructured data model (e.g., content of a text file, e-mails, web data, power point presentation etc.) (Singh and Jain, 2013). The set of automatic/semi-automatic wrappers can also be implemented for other existing data models based on the proposed TRSs. In the following section, we will explain the TRSs for the structured, semi-structured and unstructured data models .

(a) Representation of data (b) Representation of relationship

Fig. 1 Representation of data and relationship in a triple graph.

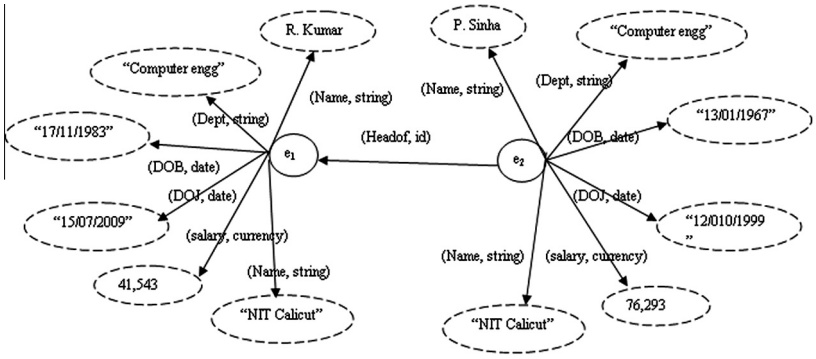


Fig. 2 A sample of triple graph.

3. Transformation Rule Sets

Algorithm 1: Decomposition Algorithm

Require: Data Model ($\mathcal{D}$)
Ensure: A bunch of triples $(\tau_1, \ldots, \tau_n)$, $n \geqslant 1$;
for each data item $\pi_i \in \mathcal{D}$ **do**
if ($C(\pi_i)$ does not exist) **then**
define a new class $C(\pi_i)$ for π_i
end if
end for
for each class $C_j(\pi_i) \in \mathcal{C}(\pi_i)$ **do**
i. Decompose the class $\mathcal{C}_j(\pi_i)$ using the function $\mathcal{R}$, where $\mathcal{R}^{\mathcal{C}_j}(\pi_i) = \{r_1^{\mathcal{C}_j}(\pi_i)$ U...U $r_m^{\mathcal{C}_j}(\pi_i)\}$, $m \geqslant 1$, where $r_k^{\mathcal{C}_j}(\pi_i)$ is a decomposition unit of class $\mathcal{C}_j(\pi_i)$;
ii. Encapsulate each decomposition unit $r_k^{\mathcal{C}_j}(\pi_i)$ into the collection of triples $(\tau_1, \ldots, \tau_r)$, $r \geqslant 1$ and $\tau_l = (\pi_i, (a_l, d_l), v_l)$, $1 \leqslant l \leqslant r$;
end for

The broader category of data models share some common properties such as underlying structure or representation of data. For example, a structured data model represents its data in the collection of tables or relations, the semi-structured data model represents its data in the form of a tree or a graph, and the unstructured data has a sequence of character or data streams or tuple streams (Sint et al., 2009). Therefore, we propose an exhaustive set of rules for structured, semi-structured, and unstructured data models based on their common properties. We have designed the decomposition algorithm (Algorithm 1) based on the decomposition theory of triple model. By applying the Algorithm 1, we present an exhaustive set of TRSs for structured, semi-structured, and unstructured data models.

3.1. Structured data model

The widely used structured data models are relational model and object relational model which organize their data into a collection of entities, and similar types of entities are grouped into an entity set or relation. Therefore, the underlying structure of the structured data models is a relation. Each relation consists of a set of tuples or records, and each tuple has a set of attributes. An attribute can be base-type or b-type (atomic attribute), row-type or r-type (molecular attribute), set-type or s-type (multi-valued attribute), and ref-type (reference attribute) (Eisenberg and Melton, 1999; Melton, 2003). A molecular attribute can be another entity or object which can further be decomposed into a set of attributes. In this way, a structured data model can be decomposed into set of relations, tuples, and attributes data items. Therefore, a structured data model consists of four data item classes which are structured database (sdb), relation, tuple, and attribute, i.e., C $(\pi) = \{sdb, relation, tuple, attribute\}$. Previously, Zhong et al. have proposed the rules for decomposing the relation data items and the tuple data items w.r.t. relational model (Zhong et al., 2008). Now, we propose the rules w.r.t. all the data item classes present in a structured data model. By applying the Algorithm 1 on the structured data model, the TRSs will be as follows:

Let us consider a structured database item (π) with name N_{sdb}, which consists of a set of relation items $\{\pi_1, \ldots, \pi_r\}$, $r \geqslant 1$. A relational item (π_i) has name N_{rel}^i and consists of s tuples $\{\pi_{i0}, \ldots, \pi_{is}\}$, $s \geqslant 0$. A tuple data item (π_{ij}) in relation (π_i) has a set of attributes $(a_1, \ldots, a_n)$, $n \geqslant 1$, and an attribute a_k has a type constructor t_k and value v_{ijk} for tuple π_{ij} in a relation π_i, $1 \leqslant i \leqslant r$, $1 \leqslant j \leqslant s$, and $1 \leqslant k \leqslant n$. An attribute ($a_k$) will be decomposed depending on its type constructor.

TRS 1: if C $(\pi) =$ sdb
$R^{sdb}(\pi) = r_1^{sdb}(\pi)$ U $r_2^{sdb}(\pi)$
TR 1.1: Name component

$r_1^{sdb}(\pi) = (\pi, (rdb\_name, string), N_{sdb})$

TR 1.2: Relation component

$r_2^{sdb}(\pi) = \{(\pi, (relation, id), \pi_1),\ldots,(\pi, (relation, id), \pi_r)\}$

TRS 2: if C (π_i) = relation

$R^{relation}(\pi_i) = r_1^{relation}(\pi_i) \cup r_2^{relation}(\pi_i)$

TR 2.1: Name Component

$r_1^{relation}(\pi_i) = (\pi_i, (relation\_name, string), N_{rel}^i)$

TR 2.2: Tuple component

$r_2^{relation}(\pi_i) = \{(\pi_i, (tuple, id), \pi_{i1}),\ldots,(\pi_i, (tuple, id), \pi_{is})\}$

TRS 3: if C (π_{ij}) = tuple

$R^{tuple}(\pi_{ij}) = r_1^{tuple}(\pi_{ij})$

TR 3.1 Attribute component

$r_1^{tuple}(\pi_{ij}) = \{(\pi_{ij},(a_1, t_1), v_{ij1}),\ldots,(\pi_{ij},(a_n, t_n), v_{ijn})\}$

TRS 4 :if C (a_k) = Attribute

Case 1: if t_k = b-type

$R^{attribute}(a_k) = r_1^{attribute}(a_k)$

TR 4.1: Attribute component

$r_1^{attribute}(a_k) = (\pi_{ij}, (a_k, d_k), v_{ijk})$

Case 2: if t_k = r-type

Let us assume that the attribute a_k has m sub-attributes $\{b_{k1},\ldots,b_{km}\}$ with data types $\{d_{k1},\ldots,d_{km}\}$. The sub-attribute b_{kl} has the value v_{ijkl} for the kth attribute of jth tuple in ith relation. Therefore, this attribute has two components: name component and sub-attribute component.

TR 4.1: Attribute component

$R^{attribute}(a_k) = r_1^{attribute}(a_k) \cup r_2^{attribute}(a_k)$

TR 4.1.1: Name component

$r_1^{attribute}(a_k) = (\pi_{ij}, (name, string), a_k)$

TR 4.1.2: Sub-attribute component

$r_2^{attribute}(a_k) = \{(\pi_{ijk}, (b_1, d_1), v_{ijk1}),\ldots,(\pi_{ijk}, (b_m, d_m), v_{ijkn})\}$

Case 3: if t_i = s-type

$R^{attribute}(a_k) = r_1^{attribute}(a_k)$

Let us assume that a multi-valued attribute (a_k) has a list of associated values $(v_{ijk1},\ldots,v_{ijkm})$ with data type d_k.

TR 4.1: Attribute component

$r_1^{attribute}(\pi_{ij}) = (\pi_{ij}, (a_k, d_k[]), \{v_{ijk1},\ldots,v_{ijkm}\})$

Case 4: if t_i = ref-type

$R^{attribute}(a_k) = r_1^{attribute}(a_k)$

Assume that an attribute a_k in one relation refers to an attribute b_j in another relation.

TR 4.1: Attribute component

$r_1^{attribute}(\pi_{ij}) = (\pi_{ij}, (a_k, id), b_j)$

Now, we take an example of a structured database, *e.g., "online book store database(OBSDB)"* as shown in Fig. 3, and decompose it into triple model using the TRSs. Our example includes the feature of a structured database, i.e., the molecular and multi-valued attributes. As shown in Fig. 3, an online book store database (OBSDB) has 5 relations with their *ids* $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$. Using TRS-1, the OBSDB database item is decomposed into name component "OBSDB" and 5 relation components $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$. Each relation has a name and consists of a number of tuples, e.g., the relation data item "author" has an *id* "π_2" and 4 tuples with *ids* $\{\pi_{21}, \pi_{22}, \pi_{23}, \pi_{24}\}$. Using TRS-2, the relation author (π_2) is decomposed into name component "author" and 4 tuple components $\{\pi_{21}, \pi_{22}, \pi_{23}, \pi_{24}\}$. Similarly, we have decomposed the other data items present in database using the proposed TRSs. Fig. 4 shows the result of decomposition of the structured database using a partial representation triple graph.

3.2. Semi-structured data model

Unlike structured data model, a semi-structured data model has a simple and flexible structure because similar kind of objects can have different structure or different number of attributes. The underlying structure of a semi-structured data is a tree or graph (Abiteboul, 1997), where each node may have a different set of attributes. The data and relationships are stored in nodes and/or edges of a tree/graph. A node is labeled with either a name or a id depending on the data model such as Object Exchange Model (OEM) with id (Abiteboul et al., 1997) and XML model with name(Clark et al., 1999). A tree/-graph based data model has two types of nodes: *Non-terminal* and *Terminal* nodes. A *non-terminal* node has a label, a set of attributes or properties, and an ordered set of children, where $children \in \{non-terminal, terminal\}$. Similarly, a *terminal* node has a label and stores the contents, i.e., a literal value. In some cases, a terminal node may have a set of attributes. Like in a file system, a file represents the terminal node which has a label, i.e., name, a set of attributes, and its contents, whereas in an xml data model, a xml text node represents a terminal node which has only a label component, i.e., tag name, and its text value, i.e., content component.

As we know, a semi-structured data model consists of non-terminal and terminal nodes. Therefore, a semi-structured data model may have non-terminal and terminal data item classes, i.e., $C(\pi) = \{non-terminal, terminal\}$. These classes can be specialized into the specific data item classes depending on the property of the respective data model. The examples of a semi-structured data are XML data, personal data, and all other data that can be modeled as a tree or graph. Fig. 5 exhibits a view of semi-structured based data model. By applying the Algorithm 1 on semi-structured data, the TRSs will be as follows:

Assume that a non-terminal node item (π_i) has a label N_{nt}, a set of attributes $\{a_{i1}^{nt},\ldots,a_{i1}^{nt}\}$ with data types $\{d_{i1}^{nt},\ldots,d_{in}^{nt}\}$, $n \geqslant 0$, and an attribute a_{ij}^{nt} has value v_{ij} for the ith node data item, where $0 \leqslant j \leqslant n$. The node item (π_i) has a set of children with $id\ \{\pi_{i1},\ldots,\pi_{im}\}$, where $m \geqslant 1$ and may vary for each node (π_i), $children \in \{non-terminal, terminal\}$. A $non-terminal$ node (π_{ij}) has number of *terminal* nodes, let p, which may vary for each node, and the kth terminal node has label N_t^k, a set of attributes (optional) with name a_{ijk}^t has data type d_{ijk}^t and value $v_{ijk}^t, 0 \leqslant k \leqslant p$, and its content is denoted as C_{ijk}.

TRS-5: if C (π_i) = $non-terminal$

$R^{ss}(\pi_i) = r_1^{ss}(\pi_i) \cup r_2^{ss}(\pi_i) \cup r_3^{ss}(\pi_i)$

TR-5.1 Label component

$r_1^{ss}(\pi_i) = (\pi_i, (label, string), N_{nt})$

TR-5.2 Attribute component

$r_2^{ss}(\pi_i) = \{(\pi_i, (a_{i1}^{nt}, d_{i1}^{nt}), v_{i1}^{nt}),\ldots,(\pi_i, (a_{in}^{nt}, d_{in}^{nt}), v_{in}^{nt})\}$

TR-5.3 Children component

$r_3^{ss}(\pi_i) = \{(\pi_i,(child, id),\pi_{i1}),\ldots,(\pi_i,(child, id),\pi_{im})\}$ where $m \geqslant 0$ and $child \in \{non-terminal, terminal\}$

TRS-6: if C (π_{ij}) = $terminal$

$R^{ss}(\pi_{ij}) = r_1^{ss}(\pi_{ij}) \cup r_2^{ss}(\pi_{ij}) \cup r_3^{ss}(\pi_{ij})$

TR-6.1 Label component

$r_1^{ss}(\pi_{ij}) = (\pi_{ij}, (label, string), N_{ij}^t)$

TR-6.2 Attribute component

$r_2^{ss}(\pi_{ij}) = \{(\pi_{ij}, (a_{ij0}^t, d_{ij0}^t), v_{ij0}^t),\ldots,(\pi_{ij}, (a_{ijp}^t, d_{ijp}^t), v_{ijp}^t)\}$, where $p \geqslant 0$

Book π_1

ISBN	Title	Price	Year	
007-124476-x	Database System Concept	450	2006	π_{11}
1-55860-4529	ORDBMS	375	2009	π_{12}
08-59256-240	Artificial Inteligent	580	2001	π_{13}

(a) Atomic attributes

Author_by π_4

ISBN	A_id	
007-124476-x	{a0101, a0102}	π_{41}
08-59256-240	{a0103}	π_{42}
1-55860-4529	{a0102}	π_{43}

(b) Multi-valued and Reference attributes

Author π_2

ID	Name		Address			e-mail	
	first_name	last_name	Street	City	Zip		
a0101	Henery	Korth	R-27	New Delhi	101011	hkorth@gmail.com	π_{21}
a0102	S	Sudarshan	S-38	New Delhi	101011	sdarshan@yahoo.com	π_{22}
a0103	Rich	Knight	E-27	Bombay	909090	rknight@rediffmail.com	π_{23}
a0104	William	Shekspear	B-26	Kanpur	210201	shekspear@gmail.com	π_{24}

(c) Molecular attributes

Published_by π_5

ISBN	P_Name	
007-124476-x	TMH	π_{51}
08-59256-240	PHI	π_{52}
1-55860-4529	PHI	π_{53}
——	——	
——	——	

(d) Reference attributes

Publisher π_3

Name	Address	Phone	URL	
Tata Mcgraw-Hill	K Place, New Delhi	{011213321, 011213322}	www.mhe.com	π_{31}
PHI	R K Puram, New Delhi	{+9111262262}	www.phi.com	π_{32}
EEE	GT Road, Meerut	{+91121123123}	www.eee.com	π_{33}

(e) Multi-valued attributes

Fig. 3 An example of structured data.

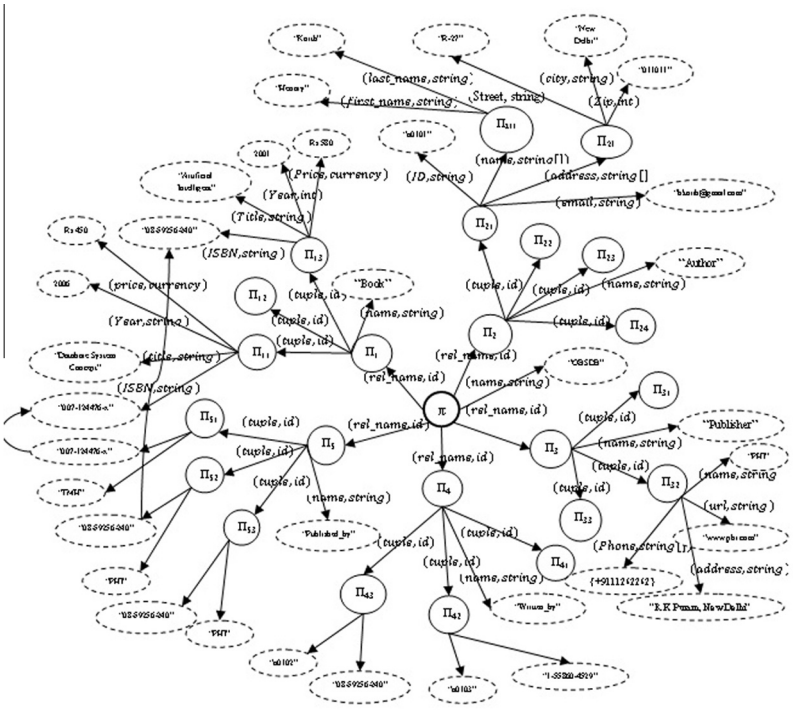


Fig. 4 A partial triple graph of structured data.

TR-6.3 Content component

$r_3^{ss}(\pi_{ij}) = (\pi_{ij}, (content, type), C_{ijk})$

Now, we empirically verify our TRSs with the help of an example of a semi-structured data shown in Fig. 6. The root node (π) has a name "books" with no attribute and two children node $\{\pi_1, \pi_2\}$. The first child (π_1) has a name "book" with an attribute "*id*" and seven leaf nodes, while second child (π_2) has a name "book" with an attribute "*id*" and six leaf nodes and so on. Fig. 6(b) illustrates the triple representation w.r.t. semi-structured shown in Fig. 6(a).

The TRSs can be applied on most of the existing semi-structured based data models such as XML data model, personal (a file-system based) data model. For example, with respect to a XML data model, the non-terminal node data item class is

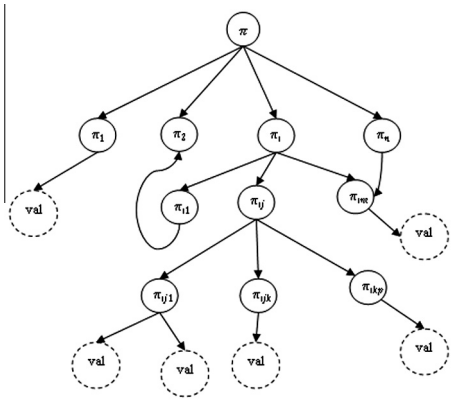


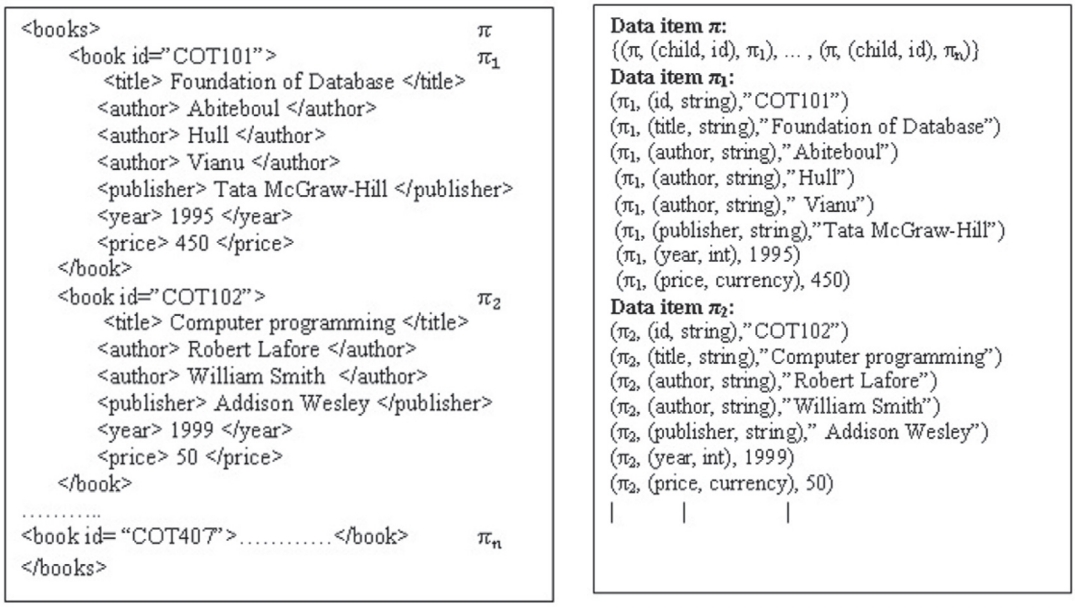
Fig. 5 A view of semi-structured data model.

xmlelement, i.e., $xmlelement \in non-terminal$, and the terminal node data item class is xmltext, i.e., $xmltext \in terminal$. Similarly, for personal data, the non-terminal node data item class is folder, i.e., $folder \in non-terminal$, and the terminal node data item class is file, i.e., $file \in terminal$.

On the other hand, we can apply the proposed algorithm on other semi-structured data models which have some distinguished properties such as iDM model (Dittrich and Salles, 2006), Interpreted Object Model (IOM) (Zhong et al., 2012), object exchange model etc. The iDM model represents the personal dataspace as a resource view graph, where each vertex is a resource view, and a resource view consists of four components: name component, tuple component, content component, and group component (Dittrich and Salles, 2006). Resource views are grouped into a resource view class, which can be decomposed into its respective components (i.e., name component, tuple component, content component, and group component). Similarly, we can apply the algorithm on Interpreted Object Model (IOM) (Zhong et al., 2012). An IOM is the newly proposed data model for PIM systems. This model represents the personal dataspace as a logical data graph, where each vertex is an Interpreted Object (IO), and each edge represents a relationship between two IOs. Therefore, the basic modeling unit in the IOM model is an IO, where each IO has unique identifier and belongs to an interpreted object class, i.e., a file, a relation, a person, an XML element etc. According to the IOM model definition, an interpreted object consists of two components: tuple or structured component and content or unstructured component. Therefore, an interpreted object class can be decomposed into tuple component and content component. The tuple component and the content component in the IOM model have same characteristics as in the iDM model. Now, we explain the TRSs for the xml data model and personal data model by extending the generic TRSs in the following sections.

3.2.1. XML data

An XML data model organizes its data, which is retrieved by using a graphical query languages (Ykhlef and Alqahtani, 2011), into a tree or a graph structure (Passi1 et al., 2002). The most commonly used XML data models are Document Object Model (DOM) (Wood et al., 1998) or XPath data model (Clark et al., 1999), which organize their contents in a tree/graph structure. The start node of an XML tree is the document node. A node in an XML tree is called xml element or tag that is identified by its name, and has zero or more attributes and a set of children nodes, $children \in \{xmlelement, xmltext\}$. A terminal node stores the value as a text, called XML text node. Whereas, the XML infoset consists of eleven information items (Cowan and Tobin, 2004; Sosnoski, 2003), but we have taken only xml element and xml text nodes into account. The TRSs for XML data model are based on the semi-structured data model due to a graph



(a) A view of Semi-Structured Data

(b) Partial representation of Semi-Structured Data

Fig. 6 An example of semi-structure data and its partial triple representation.

based structure. The nodes of a DOM or Xpath model are a document node, a root node, a set of element nodes and text nodes. Therefore, an xml data model constitutes the xmlelement node item, and the xmltext node items, i.e. C (π) = {xmlfile, xmlelement, xmltext}, where $\{xmlelement\} \in non-terminal$, and $\{xmltext\} \in terminal$ w.r.t. semi-structured data model.

Let an xml file (π) has a name N_{xfile}, a set of attributes $(a_1^{xfile}, \ldots, a_n^{xfile})$, $n \geqslant 1$; the ith attribute a_i^{xfile} has value v_i^{xfile} with data type d_i^{xfile}. The content of an XML file starts with a document node (π_{doc}), which is also an xml element node. An xml element node (π_i) has name N_E, a set of attributes $(a_{i1}^E, \ldots, a_{im}^E)$ with data type $(d_{i1}^E, \ldots, d_{im}^E)$, $m \geqslant 1; v_{ij}^E$ is the value of jth attribute of ith element, and has an ordered set of children nodes $(\pi_{i1}, \ldots, \pi_{ip})$, $p \geqslant 1, children \in \{xmlelement, xmltext\}$. Let the jth xml text node which is associated with element node "π_i" has name "N_{text}" and contents "C_{ij}". By applying the TRS-5 and TRS-6 on XML data model, the TRSs will be as follows:

TRS-A.1: if C (π) = xmlfile

$R^{xmlfile}(\pi) = r_1^{xmlfile}(\pi) \cup r_2^{xmlfile}(\pi) \cup r_3^{xmlfile}(\pi)$

TR-A.1.1: Name component

$r_1^{xmlfile}(\pi) = (\pi, (name, string), N_{xfile})$

TR-A.1.2: Attribute component

$r_2^{xmlfile}(\pi) = \{(\pi, (a_1^{xfile}, d_1^{xfile}), v_1^{xfile}), \ldots, (\pi, (a_n^{xfile}, d_n^{xfile}), v_n^{xfile})\}$

TR-A.1.3: Content component

$r_3^{xmlfile}(\pi) = (\pi, (\text{Content, id}), \pi_{doc})$

TRS A.2: if C (π_i) = xmlelement

$R^{xmlelement}(\pi_i) = r_1^{xmlelement}(\pi_i) \cup r_2^{xmlelement}(\pi_i) \cup r_3^{xmlelement}(\pi_i)$

TR A.2.1: Name component

$r_1^{xmlelement}(\pi_i) = (\pi_i, (Name, String), N_E)$

TR A.2.2: Attribute component

$r_2^{xmlelement}(\pi_i) = \{(\pi_i, (a_{i1}^E, d_{i1}^E), v_{i1}^E), \ldots, (\pi_i, (a_{im}^E, d_{im}^E), v_{im}^E)\}$

TR A.2.3: Children Component

$r_2^{xmlelement}(\pi_i) = \{(\pi_i, (children, id), \pi_{i1}), \ldots, (\pi_i, (children, id), \pi_{ip})\}$, where $p \geqslant 1, children \in \{xmlelement, xmltext\}$

TRS A.3: if C (π_{ij}) = xmltext

$R^{xmltext}(\pi_{ij}) = r_1^{xmltext}(\pi_{ij}) \cup r_2^{xmltext}(\pi_{ij})$

TR A.3.1: Name component $r_1^{xmltext}(\pi_{ij}) = (\pi_{ij}, (name, string), N_{text})$

TR A.3.2: Content component $r_1^{xmltext}(\pi_{ij}) = (\pi_{ij}, (xmltext, string), C_{ij})$

Now, we are explaining the working of TRSs with the help of a prototype example. As shown in Fig. 7, the data are stored in an xml file (π), named "Bookstore.xml". Let the content of a file starts with a document node (π_{doc}). The document node has two children node prolog (π_{pro}) and root node (π_{root}), named "Bookstore". The root node has six children element nodes {π_1 - π_6}. Each element has a name "book" with an attribute "id", and a number of children nodes; the "author" element within "book" element node is further decomposed into xml text nodes with name "first_name", "last_name", and "email". Fig. 7 illustrates a view of xml data and their partial triple representation.

3.2.2. Personal data

The data related to a person, stored in his desktop with possible extension of mobile device, e-mail, USB drive etc, is called personal data. In general, the underlying structure of the personal data is a tree or a graph which includes the files and folders (Dittrich et al., 2007; Zhong et al., 2012). With respect to the semi-structured data, a folder represents the non-terminal nodes, and a file represents the terminal nodes. Meanwhile, a folder has a name, a set of attributes, and a set of children nodes, where $children \in \{file, folder\}$, and a file has a name, a set of attributes, and its contents either unstructured or semi-structured. Therefore, a personal data model constitutes file and folder data item classes, i.e., C (π) = {file, folder}, where $file \in terminal$ and $folder \in non-terminal$ w.r.t. the semi-structured data item classes. A folder data item class will be decomposed into its name component, attribute component and children component, and a file data item class will be decomposed into name component, attribute component and content component.

Assume that a folder data item (π_i), ($i \geqslant 1$) has name N_{folder}, a set of attributes $(a_1^{folder}, \ldots, a_n^{folder})$, with data types $(d_1^{folder}, \ldots, d_n^{folder})$, $n \geqslant 1$, and an attribute a_j^{folder} has value v_{ij}^{folder} for the ith data item, and has m number of children with id $(\pi_{i1}, \ldots, \pi_{im})$, $m \geqslant 1$. A file data item (π_i) has name N_{file}, a set of attributes $(a_1^{file}, \ldots, a_n^{file})$, with data types $(d_1^{file}, \ldots, d_n^{file})$ and values corresponding to these attributes are $(v_1^{file}, \ldots, v_n^{file})$ respectively, and content of the file is C_{file}. By applying the TRS-5 and TRS-6 on personal data, the TRSs will be as follows:

TRS B.1: if C(π_i)= folder

$R^{folder}(\pi_i) = r_1^{folder}(\pi_i) \cup r_2^{folder}(\pi_i) \cup r_3^{folder}(\pi_i)$

TR B.1.1: Name component

$r_1^{folder}(\pi_i) = (\pi_i, (name, string), N_{folder})$

TR B.1.2: Attribute component

$r_2^{folder}(\pi_i) = \{(\pi_i, (a_1^{folder}, d_1^{folder}), v_1^{folder}), \ldots, (\pi_i, (a_n^{folder}, d_n^{folder}), v_n^{folder})\}$

TR B.1.3: Children Component

$r_3^{folder}(\pi_i) = \{(\pi_i, (child, id), \pi_{i1}), \ldots, (\pi_i, (child, \text{id}), \pi_{im})\}$, where, $child \in$ {folder, file}

TRS B.2: if C (π_i)= file

$R^{file}(\pi_i) = r_1^{file}(\pi_i) \cup r_2^{file}(\pi_i)$

TR B.2.1: Name component

$r_1^{file}(\pi_i) = (\pi_i, (name, string), N_{file})$

TR B.2.2: Attribute Component

$r_1^{file}(\pi_i) = \{(\pi_i, (a_1^{file}, d_1^{file}), v_1^{file}), \ldots, (\pi_i, (a_n^{file}, d_n^{file}), v_n^{file})\}$

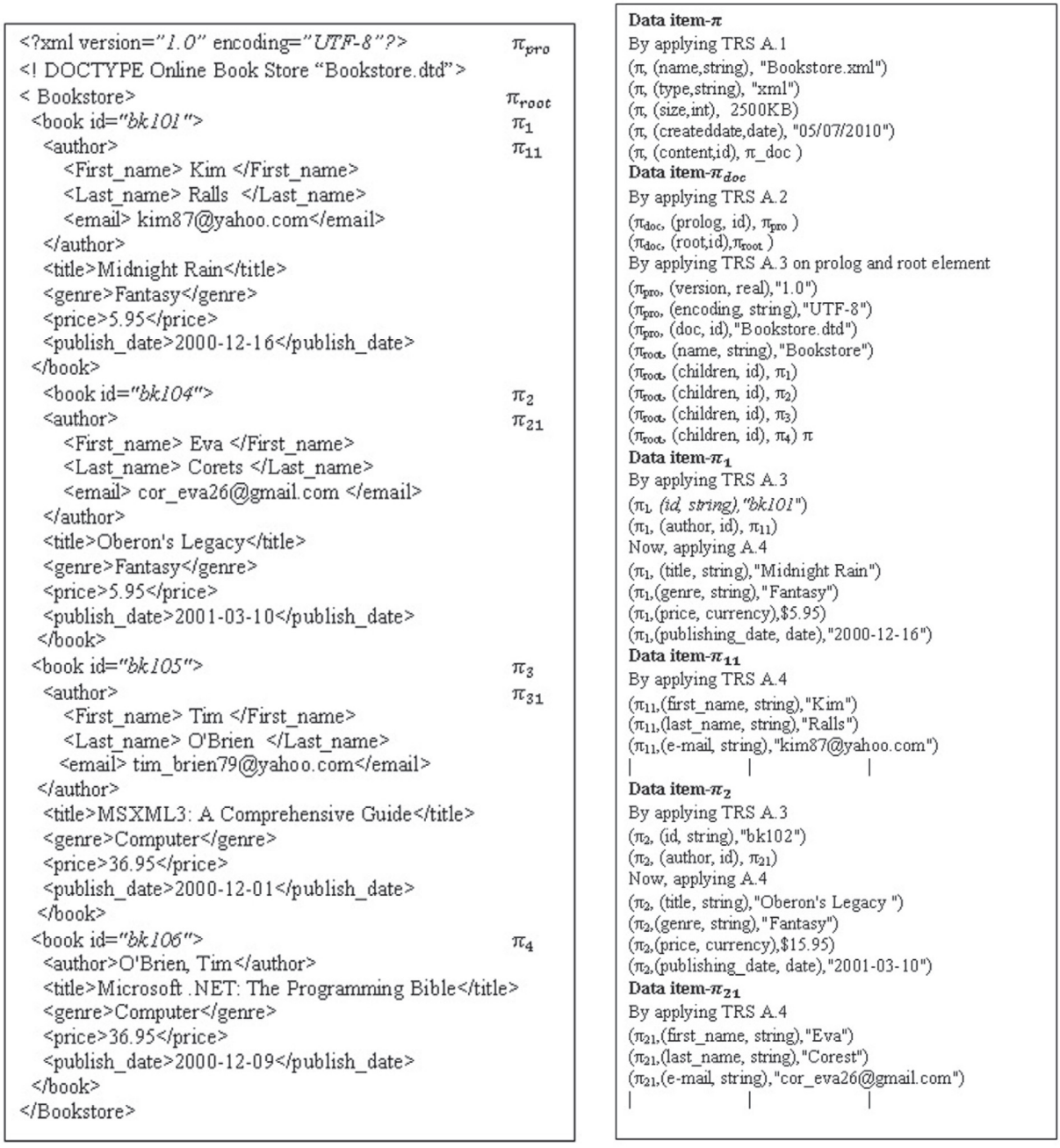
TR B.2.3: Content Component

$r_2^{file}(\pi_i) = (\pi_i, (content, \text{string}), C_{file})$

An example of a personal data (a file/folder hierarchy) and corresponding partial triple graph is shown in Fig. 8. The triple model fulfills the gap between the inside vs. outside data in a file system by representing them through a single graph. Therefore, a user can uniformly retrieve these data using a single query language. The content of a file can be explicitly organized into a tree/graph like structure depending on their content type. Here, we consider the content of a file as a single data unit. We discuss the TRSs for decomposing the content of a file in the next section.

3.3. Unstructured data model

An unstructured data have no predefined data model, and is treated as a sequence of data(or tuple) streams (Dittrich and Salles, 2006). A prominent kind of unstructured data includes



(a) A view of XML data

(b) A triple representation of XML data

Fig. 7 An example of XML data.

text data, e-mail, web data, multimedia data such as audio, video, image, graphics etc. An unstructured document consists of data segments. A *data segment* may contain other data segments and/or data elements, which can be organized as a tree/graph structure explicitly (Buneman et al., 1996; Buneman et al., 1997). A *data element* is the smallest unit of data in a data segment. For example, in a business document, an order information, an invoice information, and a shipping information form data segments, while an order number, an invoice number, a per unit cost, and an order date are data elements. Therefore, an unstructured data models can be decomposed into a collection of data segments and each data segment is decomposed into a collection of data segments and/or data elements, then each data element is encapsulated into a triple, which has an unique identifier. For example, the data element "Mr Beans is a member of an organization X" will be decomposed as (*id*, (*name, string*), "*Mr. Beans*") and $(id, (isMemebrof, string), organizationX)$.

In some cases, a data segment may have a unique name, e.g., in case of an article, which has a number of data segments such as title, sections etc. Each section forms a data segment and has a unique name, but the paragraphs in a section do

not have any name even though they also form a data segment. Therefore, an unstructured data model constitutes document file, data segment, and data element classes, i.e., $C(\pi) = \{documentfile, datasegment, dataelement\}$. A view of unstructured data representation is shown in Fig. 9. By applying the Algorithm 1 on the unstructured data model, the TRSs will be as follows:

Let us assume that an unstructured document file (π) has name N_{file}, a set of attributes $(a_1, \ldots, a_n)$, and the attribute a_j has value v_j with type d_j, where $0 \leqslant j \leqslant n$ and $n \geqslant 0$. The content of the document file is represented by (π_{un}), which is further decomposed into a number of data segments $(\pi_1, \ldots, \pi_m)$, where $m \geqslant 0$. Let a data segment (π_i) has name N_i^{seg}, which may be null, and a number of children $(\pi_{i1}, \ldots, \pi_{ip})$, where $p \geqslant 0$ and $children \in \{datasegment, dataelement\}$. A data element ($\pi_{ij}$) will be decomposed into r units with label l_{ijk} and value v_{ijk} with data type d_{ijk}, where $0 \leqslant k \leqslant r$.

TRS 7: if $C(\pi) =$ documentfile

$R^{documentfile}(\pi) = r_1^{documentfile}(\pi) \cup r_2^{documentfile}(\pi) \cup r_3^{documentfile}(\pi)$

TR 7.1: Name Component

$r_1^{documentfile}(\pi) = (\pi, (name, string), N_{file})$

TR 7.2: Attribute Component

$r_1^{documentfile}(\pi) = \{(\pi,(a_1, d_1), v_1),\ldots,(\pi,(a_n, d_n), v_n)\}$

TR 7.3: Content Component

$r_1^{documentfile}(\pi) = (\pi, (content, id), \pi_{un})$

TRS 8: if $C(\pi_i) =$ datasegment

$R^{datasegment}(\pi_i) = r_1^{datasegment}(\pi_i) \cup r_2^{datasegment}(\pi_i)$

TR 8.1: Name Component

$r_1^{datasegment}(\pi_i) = (\pi_i, (name, string), N_i^{seg})$

TR 8.2: Children Component

$r_2^{datasegment}(\pi_i) = \{(\pi_i, (children, id), \pi_{i1}),\ldots,(\pi_i, (children, id), \pi_{ip})\}$

TRS 9: if $C(\pi_{ij}) =$ dataelement

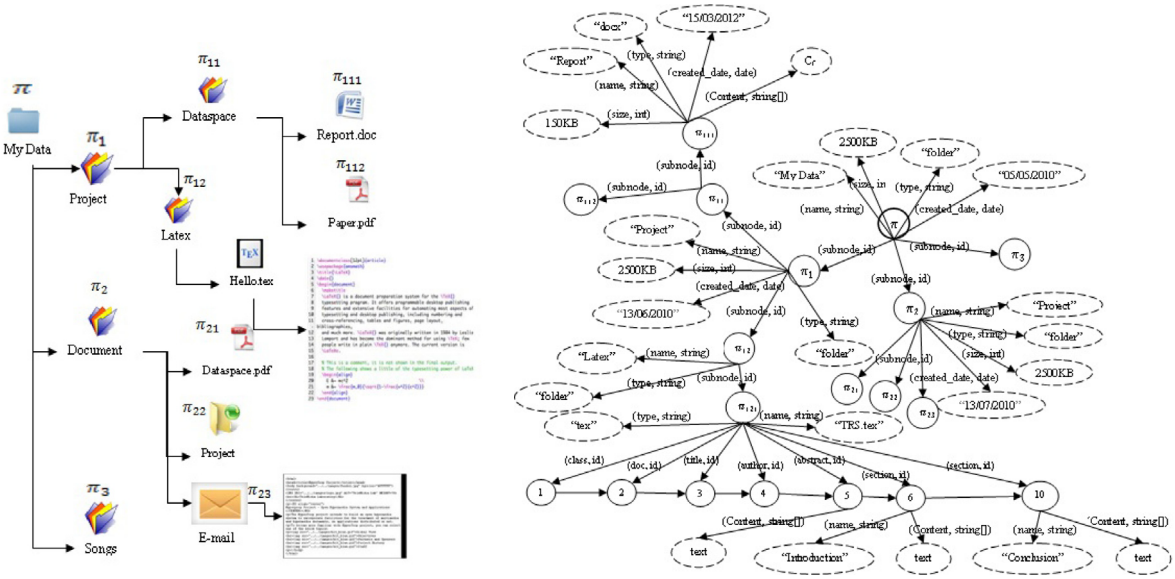
$R^{dataelement}(\pi_{ij}) = r_1^{dataelement}(\pi_{ij})$

TR 9.1: Content Component

$r_1^{dataelement}(\pi_{ij}) = \{(\pi_{ij}, (l_{ij1}, d_{ij1}), v_{ij1}),\ldots,(\pi_{ij}, (l_{ijr}, d_{ijr}), v_{ijr})\}$

The proposed TRSs for unstructured data are simple and straight forward. Unlike Information Extraction (IE) tool, we translate the unstructured data into the collection of triples without extracting the structure from the data (Grishman, 1997; Doan et al., 2009a; Doan et al., 2009b; Al-Mathami, 1998), because the existing IE tools have the following disadvantages (Kastrati et al., 2011): *first,* such approaches are costly due to a very large collection of data have high preprocessing cost, *second,* automatic extraction of structure is a source of uncertainty (Sarma et al., 2009), and *third,* they consist of out-of-dated version of extracted data already stored in somewhere. Therefore, we have adopted an approach proposed by F. Kastrati et. al. (Kastrati et al., 2011), which extracts the structure from unstructured data on-the-fly, and processes the query just-in-time. F. Kastrati et. al. have proposed a system which supports the structured queries on unstructured data by identifying the relationships among the plain text without a ''global schema'' or ''up-front efforts''.

The World Wide Web is a good example of unstructured data containing enormous load of information that is often embedded in plain text, images, audio/vedio etc. However, the information on the web may get updated frequently and have different meaning from different user perspective. Therefore, an IE based approach is not suitable for data extraction. In this work, we have adopted a just-in-time query processing over a large collection of documents, which are result of a corpus selection procedure (Kastrati et al., 2011). This approach utilizes the functionality of search engine for selecting a relevant document based on the input keywords, and locating the appropriate data segments from a selected document. Each data segment is decomposed into a set of data elements, and each data element is encapsulated into a triple. There are lots of works that have been cited in literature which extract the structured information from the text data on-the-fly (Liu et al., 2006; Chu et al., 2007; Yang et al., 2013).



(a) A view of personal data

(b) A part of triple graph of Personal Data

Fig. 8 An example of personal data.

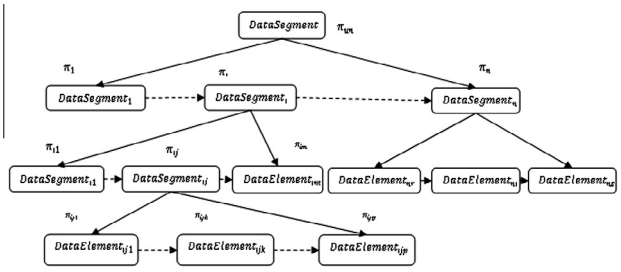


Fig. 9 Representation of unstructured data.

4. Comparison

In this section, we compare our work with the existing work (Zhong et al., 2008) as shown in Table 1. From Table 1, we concluded that the proposed rules are exhaustive, and cover the wide range of data models while the existing rules are limited and specific for a few data models. In this paper, we have addressed the rules for the structured, semi-structured, and unstructured data models.

Our TRSs for structured data models support the relational as well as object relational data models because they included the decomposing of the molecular as well as multi-valued attributes while the existing rules were not applicable on the object relation data models. In this work, we have considered the decomposition of a database, relation, tuple and attribute data items while the existing work considered only the decomposition of a relation and a tuple data item. We have verified our rules for an object relational (i.e., a structured) database shown in Fig. 3. We have also implemented the wrapper based on the proposed TRSs for structured data. Our wrapper is automatic, and is independent from the underlying structure of a database. The implementation of wrapper has been uploaded in our web site (Singh and Jain, 2013). Our wrappers fulfilled the requirement of a dataspace system, and performed in a pay-as-you-go manner.

On the other hand, the proposed TRSs for semi-structured data are not specific for a single data model but they can be applied on most of the tree/graph based data models depending on their respective properties. We have extended our rules for an XML and a file system based data model. Previously, the authors proposed the rules for the file/folder hierarchy, iDM, and XML data (Zhong et al., 2008). With respect to xml data, they considered the content of an xml file like a simple text data, and decomposed them like the content of a text file, while in this work, we have considered the content of an xml file like a tree structure, and given the TRSs for them. On the other hand, due to flexible structure of a semi-structured data model, the proposed TRSs may not be applicable for all the graph-based data models. In such cases, we can apply the proposed algorithm for decomposing such data model such as iDM model and IOM model as in Section 3.2. We have implemented a set of fully automatic wrappers for the personal data, XML data, and latex data etc. Our wrappers for semi-structured data models are available in our web site (Singh and Jain, 2013).

In the existing work (Zhong et al., 2008), Zhong et al. have not defined the rules for decomposing the unstructured data explicit, while we have proposed TRSs for unstructured data model which are applicable over the most of the existing unstructured data like text data, web data, e-mail data, multimedia data etc. The proposed rules are simple and straight forward. They are not based on the IE based approach while they extract the structure "on-the-fly" based on a user query. The automatic extraction of information may cause a source of uncertainty which can be improved using a fuzzy logic based mechanism (Hamani et al., 2014; Mukherjee and Kar, 2012). We have manually implemented a set of wrapper for the variety of text data, e-mail data and web data which is not an efficient way for a dataspace system. The implementation of the fully automatic wrappers for unstructured data is not easy due to their undetermined structure which can be determined either manually or using some machine learning approach. The development of automatic wrapper for web data is not easier due to the diversity of the structure. We have uploaded the implemented wrappers for few unstructured data in our web site (Singh and Jain, 2013).

Table 1 Comparison between the proposed TRSs with existing TRSs.

Data models	TRSs		
	Different type of data models	Existing TRSs	Proposed TRSs
Structured	Relational	✓	✓
	Object relational	*X*	✓
Semi-structured	File/folder	✓	✓
	XML	✓	✓
	Other semi-structured model	X	✓
Unstructured	Text	*X*	✓
	E-mail	*X*	✓
	Web	X	✓
	Other unstructured data model	X	✓

5. Discussion

In this section, we made a discussion about our work and advocate that *First,* the triple model has promising structure for representing the heterogeneous data in the dataspace systems. *Second,* a newly added data model can be easily adopted by the dataspace system. *Third,* there is no chance of uncertainty at data and schema level, and *Finally,* the triple model supports the simple graph based query language for efficient retrieval of data from the dataspace without resolving the semantic heterogeneity. Now, we elaborate the meaning and importance of each point successively.

The triple model is a semi-structured based data model which can easily incorporate the structured, semi-structured, and unstructured based data models in its core. This model has a simple and flexible structure. Unlike the iDM model (Dittrich and Salles, 2006), the triple model stores the data and relationships on the nodes and edges of the graph respectively. One can easily extract the data from the dataspace using a simple graph based query language. Therefore, the triple model is a suitable candidate for the uniform representation of heterogeneous data in the dataspace.

Due to exponential growth of the data and database management systems, there is a possibility of adding a new data

model by the data management communities. Therefore, the newly added model should be easily accepted by the dataspace system. Our decomposition algorithm can be easily applied on the newly added data models.

On the other hand, when the data get transformed from one format to another format, there is the chance of uncertainty at various levels (Sarma et al., 2009). In contrast to triple model, the transformation process is based on the predefined transformation rules. Therefore, there is no chance of uncertainty at data or schema level. Still, there is a requirement for addressing the uncertainty at query level because a user query (i.e., a simple keyword query) may get translated into a graph-based query explicitly (Sarma et al., 2009).

The heterogeneities present among the data can be classified into structural heterogeneity, syntactic heterogeneity, and semantic heterogeneity (Wache et al., 2001). *Structural heterogeneity* is due to difference among the structure or schema of the same data. *Syntactic heterogeneity* is also called technical heterogeneity. This is present among data because different data sources may use different data models or data management systems to manage their data. *Semantic heterogeneity* is present due to difference in the content of data and their intended meanings. A dataspace consists of highly semantically diverse data coming from different data sources. The triple model deals with structural and syntactic heterogeneities, and positively bridges the structural and technological gaps present among the data. On the other hand, the triple model does not fulfill the semantic gaps among the data in dataspace. Therefore, there is requirement of addressing the semantic heterogeneity from dataspace in pay-as-you-go fashion. Previously, the researchers have proposed the various approaches for dataspace system based on the user feedbacks (Belhajjame et al., 2013; Belhajjame et al., 2011; Belhajjame et al., 2010; Doan and McCann, 2003; Jeffery et al., 2008). In contrast to a dataspace system, the processing of user feedback should be as automatic as possible.

6. Conclusion and future direction

In this work, we have designed an algorithm based on the decomposition theory of the triple model, and proposed a set of transformation rules for structured, semi-structured, and unstructured data models. Our algorithm can be applicable over most of the existing data models and easily able to incorporate a newly added data model into the dataspace, this is the beauty of our decomposition algorithm. We have empirically verified the proposed rules on varieties of existing data models like relational model, object relational model, XML model, personal data model, and text data model, and conclude that the proposed rules are applicable over the wide range of the heterogeneous data. The rules can further be extended for other kind of data such as multimedia data, web data by applying the proposed TRSs. On the other hand, one can create a conceptual model by finding the semantically equivalent schema elements in dataspace using "from-data-to-schema" approach.

Acknowledgments

We are very thankful to Dr. V K Panchal (Scientist-E and Associate Director, DTRL lab, Defence Research and Development Organisation (DRDO), New Delhi, India) for providing the valuable suggestions, and his guidance to prepare this paper. His suggestions help us to improve the paper more efficiently and understandable manner.

References

Abiteboul, S., 1997. Querying semi-structured data. In: Database Theory ICDT'97. Springer, pp. 1–18.

Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.L., 1997. The lorel query language for semistructured data. Int. J. Digit. Lib. 1 (1), 68–88.

Al-Mathami, S.S., 1998. Knowledge discovery in databases: a query-guided approach. J. King Saud Univ. – Comput. Inf. Sci. 10, 15–25.

Belhajjame, K., Paton, N., Fernandes, A., Hedeler, C., Embury, S., 2011. User feedback as a first class citizen in information integration systems. In: Biennial Conference on Innovative Data Systems Research, pp. 175–183.

Belhajjame, K., Paton, N.W., Embury, S.M., Fernandes, A.A., Hedeler, C., 2010. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In: Proceedings of the 13th International Conference on Extending Database Technology. ACM, pp. 573–584.

Belhajjame, K., Paton, N.W., Embury, S.M., Fernandes, A.A., Hedeler, C., 2013. Incrementally improving dataspaces based on user feedback. Inf. Syst., 656–687

Bizer, C., Heath, T., Berners-Lee, T., 2009. Linked data-the story so far. Int. J. Semantic Web Inf. Syst. (IJSWIS) 5 (3), 1–22.

Buneman, P., Davidson, S., Fernandez, M., Suciu, D., 1997. Adding structure to unstructured data. In: Database Theory ICDT'97, pp. 336–350.

Buneman, P., Davidson, S., Hillebrand, G., Suciu, D., 1996. A query language and optimization techniques for unstructured data. ACM SIGMOD Rec. 25 (2), 505–516.

Chu, E., Baid, A., Chen, T., Doan, A., Naughton, J., 2007. A relational approach to incrementally extracting and querying structure in unstructured data. In: Proceedings of the 33rd International Conference on Very large Data Bases. VLDB Endowment, pp. 1045–1056.

Clark, J., DeRose, S., et al., 1999. Xml path language (xpath).

Cowan, J., Tobin, R., 2004. Xml information set.

Dessì, N., Pes, B., 2009. Towards scientific dataspaces. In: IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies. WI-IAT'09, vol. 3. IET, pp. 575–578.

Dittrich, J., Salles, M., 2006. iDM: a unified and versatile data model for personal dataspace management. In: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB Endowment, pp. 367–378.

Dittrich, J.-P., Blunschi, L., Färber, M., Girard, O.R., Karakashian, S.K., Salles, M.A.V., 2007. From personal desktops to personal dataspaces: a report on building the imemex personal dataspace management system. In: BTW, pp. 292–308.

Dittrich, J.-P., Salles, M., Karaksashian, S., 2006. imemex: a platform for personal dataspace management. In: SIGIR PIM Workshop, pp. 22–29.

Doan, A., McCann, R., 2003. Building data integration systems: a mass collaboration approach. In: International Workshop on Web and Databases, pp. 183–188.

Doan, A., Naughton, J., Baid, A., Chai, X., Chen, F., Chen, T., Chu, E., DeRose, P., Gao, B., Gokhale, C., et al., 2009. The case for a structured approach to managing unstructured data. arXiv preprint <arXiv:0909.1783>.

Doan, A., Naughton, J.F., Ramakrishnan, R., Baid, A., Chai, X., Chen, F., Chen, T., Chu, E., DeRose, P., Gao, B., et al, 2009b. Information extraction challenges in managing unstructured data. ACM SIGMOD Rec. 37 (4), 14–20.

Dong, X., Halevy, A., Yu, C., 2009. Data integration with uncertainty. VLDB J. 18 (2), 469–500.
Eisenberg, A., Melton, J., 1999. Sql: 1999, formerly known as sql3. ACM Sigmod Rec. 28 (1), 131–138.
El-Sappagh, S.H.A., Hendawi, A.M.A., El Bastawissy, A.H., 2011. A proposed model for data warehouse etl processes. J. King Saud Univ.–Comput. Inf. Sci. 23 (2), 91–104.
Elsayed, I., Brezany, P., 2010. Towards large-scale scientific dataspaces for e-science applications. In: Database Systems for Advanced Applications. Springer, pp. 69–80.
Franklin, M., 2009. Dataspaces: progress and prospects. Dataspace: the final frontier. LNCS 5588, 1–3.
Franklin, M., Halevy, A., Maier, D., 2005. From databases to dataspaces: a new abstraction for information management. ACM Sigmod Rec. 34 (4), 27–33.
Grishman, R., 1997. Information extraction: techniques and challenges. In: Information Extraction A Multidisciplinary Approach to an Emerging Information Technology. Springer, pp. 10–27.
Halevy, A., Franklin, M., Maier, D., 2006. Principles of dataspace systems. In: Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. ACM, pp. 1–9.
Hamani, M.S., Maamri, R., Kissoum, Y., Sedrati, M., 2014. Unexpected rules using a conceptual distance based on fuzzy ontology. J. King Saud Univ.–Comput. Inf. Sci. 26 (1), 99–109.
Hedeler, C., Belhajjame, K., Fernandes, A.A., Embury, S.M., Paton, N.W., 2009. Dimensions of dataspaces. In: Dataspace: the final frontier. LNCS, vol. 5588. Springer, pp. 55–66.
Jeffery, S., Franklin, M., Halevy, A., 2008. Pay-as-you-go user feedback for dataspace systems. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. ACM, pp. 847–860.
Kastrati, F., Li, X, Quix, C., Khelghati, M., 2011. Enabling structured queries over unstructured documents. In: 2011 12th IEEE International Conference on Mobile Data Management (MDM), vol. 2. IEEE, pp. 80–85.
Lenzerini, M., 2002. Data integration: a theoretical perspective. In: Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. ACM, pp. 233–246.
Liu, J., Dong, X., Halevy, A.Y., 2006. Answering structured queries on unstructured data. In: WebDB, vol. 6. Citeseer, pp. 25–30.
Melton, J., 2003. Advanced SQL, 1999: Understanding Object-relational and Other Advanced Features. Morgan Kaufmann Pub.
Mirza, H., Chen, L., Chen, G., 2010. Practicability of dataspace systems. JDCTA 4, 233–243.
Mukherjee, S., Kar, S., 2012. Application of fuzzy mathematics and grey systems in education. J. King Saud Univ.–Comput. Inf. Sci. 24 (2), 157–163.
Ngomo, A.-C.N., 2012. On link discovery using a hybrid approach. J. Data Semantics 1 (4), 203–217.
Passi1, K., Lane, L., Madria, S., Sakamuri, B., Mohania, M., Bhowmick, S., 2002. A model for xml schema integration. In: E-Commerce and Web Technologies, pp. 193–202.
Sarma, A., Dong, X., Halevy, A., 2009. Data modeling in dataspace support platforms. Conceptual Modeling: foundations and applications, pp. 122–138.
Singh, M., Jain, S.K., 2011. A survey on dataspace. Adv. Network Security Appl., 608–621
Singh, M., Jain, S.K., 2013. Wrappers for the dataspace system. < http://mrtnmrt.hpage.com/ > .
Sint, R., Schaffert, S., Stroka, S., Ferstl, R., 2009. Combining unstructured, fully structured and semi-structured information in semantic wikis. In: Fourth Workshop on Semantic Wikis, ESWC2009.
Sosnoski, D.M., 2003. Xbis xml infoset encoding. In: W3C Workshop on Binary Interchange of XML Information Item Sets, World Wide Web Consortium.
Van Hage, W.R., van Erp, M., Malaisé, V., 2012. Linked open piracy: a story about e-science, linked data, and statistics. J. Data Semantics 1 (3), 187–201.
Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S., 2001. Ontology-based integration of information-a survey of existing approaches. In: IJCAI-01 Workshop: Ontologies and Information Sharing, Vol. 2001. Citeseer, pp. 108–117.
Wood, L., Le Hors, A., Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Nicol, G., Robie, J., Sutor, R., et al., 1998. Document object model (dom) level 1 specification, W3C Recommendation 1.
Yang, D., Shen, D.-R., Yu, G., Kou, Y., Nie, T.-Z., 2013. Query intent disambiguation of keyword-based semantic entity search in dataspaces. J. Comput. Sci. Technol. 28 (2), 382–393.
Ykhlef, M., Alqahtani, S., 2011. A survey of graphical query languages for xml data. J. King Saud Univ. – Comput. Inf. Sci. 23 (2), 59–70.
Zhong, M., Liu, M., Chen, Q., 2008. Modeling heterogeneous data in dataspace. In: IEEE International Conference on Information Reuse and Integration, IRI 2008, pp. 404–409.
Zhong, M., Liu, M., He, Y., 2012. 3sepias: a semi-structured search engine for personal information in dataspace system. Inf. Sci. 218, 31–50.