Rule-Based Wrappers for a Dataspace System

Mrityunjay Singh, Niranjan Lal and Shashank Yadav

Abstract: ***Many organizations/individuals face the problem of managing a large amount of distributed and heterogeneous data in an efficient manner. The dataspace technology addresses this problem in an efficient manner. A dataspace system is a new abstraction for integrating heterogeneous data sources distributed over the sites that offers on-demand data integration solution with less effort and provides an integrated way of searching & querying capability over heterogeneous data sources. We require the set of automatic wrappers to extract the desired data from their data sources. A wrapper extracts the requested data from their respective data sources, and populates them into the dataspace in desired format (e.g., triple formate).***

This work presents a set of rule-based wrappers for a dataspace system that wrappers operate in "pay-as-you-go" manner. We have divided our work into two parts: discussing a set of Transformation Rules (TRSs) and designing of a set of wrappers based on the TRSs. First, we explain the working of the TRSs for structured, semi-structured, and unstructured data model, then, we discuss the designing of rule-based wrappers for dataspace system based on TRSs. We have successfully implemented the wrapper for some real and synthetic data sets. Our some of the wrappers are semi-automatic because they requires the human involvement during the data extraction and translation.

Index Terms**: *Dataspace System; Triple Model; Rule-based Wrappers; TripletDS; Pay-as-you-go***

I. INTRODUCTION

In recent past, the attention has been drawn on effectively managing of distributed and heterogeneous data [29, 24], and excellently solved by the dataspace system [22]. With respect to the other existing data integration systems, a dataspace system provides a powerful mechanism for searching and querying the mix of heterogeneous data in an integrated manner, and offer best effort answers to the users before providing a complete semantic mapping among the data [23]. A dataspace system has the distinguish properties
from these systems [32]. Therefore, the invention of a dataspace system requires a new data model and a new query language for efficiently representing and retrieving the heterogeneous the data in dataspace. There are several model has been cited in literature [20, 37, 44, 42, 28, 45, 31, 36, 33, 27]. and adopted by the various dataspace projects [25, 41, 21, 30, 26, 17, 45]. The basis data unit of most of these existing data models are either triple or resource view [20]. The iDM model is less suitable due to the restructuring philosophy and its complex query language (iQL) [37, 44], and triple model has simple and flexible structure.

Revised Manuscript Received on December 22, 2018.

Mrityunjay Singh, Department of Computer Science and Engineering, Bundelkhand Institute of Engineering and Technology, Jhansi- 284001, India

Niranjan Lal, Department of Computer Science and Engineering, SET, Mody University, Sikar - 332311, India

Shashank Yadav, Department of Information Technology, SRM IST, NCR Campus, Modinagar, 201204, India

In 2008, Zhong et. al. have modified and extended the RDF model, called triple model, for representing the heterogeneous data. According to them, the triple model is based on the decomposition philosophy and represents the variety of heterogeneous data without losing their semantics [44]. In the triple model, a triple (T) is the smallest unit of data which is represented as 3-tuples (S;P;O). Each component of triple is defined as follows: the subject (S) is an integer, which is a unique identifier of a data item. The predicate (P) is a 2-tuple (l;d), where label l is a finite string represents the name of property of triple, and d is also a finite string that represent the data type of the property (p). The object (O) is an byte array which store the actual data. It is different from the RDF triple in following ways: the subject component of triple model is an integer value instead of URI string in RDF model. The predicate component of the triple
model is 2-tuple which stores the metadata of triple, which store the semantics of data. We explore the main difference between the RDF model [8] and triple model [44] with the help of a simple example in the next Section (c.f. Section 2). The most important aspect of the triple model is that it transforms the data as they are without altering their data and semantics. Due to distinguish property of the triple model w.r.t. RDF model, there is the requirement of a set transformation rules for transforming the variety of data into triple, and a set of automatic rule based wrappers which extract the data from their respective data sources and by applying the transformation rules convert them into triple.

In this work, we extend the previous work [44], and propose the exhaustive set of transformation rules for structured, semi-structured, and unstructured data. We have analyzed the structure of existing kinds of data such as structured, semi-structured and unstructured data and propose the set of transformation rules with respect to them.
Then, we design a core wrapper w.r.t. a dataspace system. Our core wrapper works in two phase: Extraction phase and transformation phase. In extraction phase, the wrapper extracts the data as well as underlying structure of the data from data source, and in transformation phase, the wrapper apply the transformation rules on the extracted data, and translate them into the collection of triple based on their underlying structure. We verify our core wrapper module by implementing the set of automatic rule based wrappers for the variety of data.
We have implemented the set of automatic and rule-based wrappers for the various structured, semi-structured and unstructured data, and found that our wrappers fulfill the requirement of the dataspace systems by extracting the data from the data sources in "pay-as-you-go" manner.

PublicationDB (π)

Persons π_1

PersonID	name	email	
1	M. Franklin	franklin@cs.berklin.edu	π_{11}
2	Xin Dong	lunadong@cs.washington.edu	π_{12}

(a) Persons Relation

Papers π_2

PaperID	Title	
11733836	Principles of Dataspace Systems	π_{21}
1247487	Indexing Dataspaces	π_{22}

(b) Papers Relation

authorBy π_3

PaperID	PersonID	
11733836	1	π_{31}
1247487	2	π_{32}

(c) authorBy Relation

Figure 1: Example Data Set

```
@base <http://xyz.example/DB/>.
@prefix xsd: <http://www.w3c.org/2001/XMLSchema#>.
<Persons/PersonID=1>    rdf:type          <Persons>
<Persons/PersonID=1>    <Persons#PersonID>  1
<Persons/PersonID=1>    <Persons#name>      M. Franklin
<Persons/PersonID=1>    <Persons#email>     franklin@cs.berklin.edu
<Persons/PersonID=2>    rdf:type          <Person>
<Persons/PersonID=2>    <Persons#PersonID>  2
<Persons/PersonID=2>    <Persons#name>      Xin Ding
<Persons/PersonID=2>    <Persons#email>     lunadong@cs.washington.edu

<Papers/PaperID=11733836>  rdf:type         <Papers>
<Papers/PaperID=11733836>  <Papers#PaperID> 11733836
<Papers/PaperID=11733836>  <Papers#Title>   Principles of Dataspace Systems
<Papers/PaperID=1247487>   rdf:type         <Papers>
<Papers/PaperID=1247487>   <Papers#PaperID> 1247487
<Papers/PaperID=1247487>   <Papers#Title>   Indexing Dataspaces
```

(a) Representation in RDF Model

Subject	Predicate	Object
π_1	(rel_name , String)	Persons
π_1	(tuple, id)	π_{11}
π_1	(tuple, id)	π_{12}
π_{11}	(PersonID, integer)	1
π_{11}	(name, String)	M. Franklin
π_{11}	(email, string)	franklin@cs.berklin.edu
π_{12}	(PersonID, integer)	2
π_{12}	(name, String)	Xin Ding
π_{12}	(email, string)	lunadong@cs.washington.edu
π_2	(rel_name , String)	Papers
π_2	(tuple, id)	π_{21}
π_2	(tuple, id)	π_{22}
π_{21}	(PaperID, integer)	11733836
π_{21}	(Title, String)	Principles of Dataspace Systems
π_{22}	(PaperID, integer)	1247487
π_{22}	(Title, String)	Indexing Dataspaces

(b) Representation in Triple Model

Figure 2: Representation of relational database

We have applied our wrappers on various real and synthetic data sets for their verification and evaluate the correctness of the proposed TRSs. Our few wrappers are semi-automatic because they require the human involvement during the data extraction and transformation. Therefore, the contributions of this paper are two folds:

- We discuss a set of Transformation Rules for converting structured, semi-structured and unstructured data into triple data as discussed in our previous work [38].
- We discuss the design and implementation of a set automatic/semi-automatic rule-based wrappers for the variety of data.

The rest of paper is organized as follows: Section 2 presents basis of triple model. Section 3 discuss the TRSs for structured, semi-structured, and unstructured data, and Section 4 presents the rule based wrappers w.r.t. them. The experimental evaluation of the work is presented in Section 5, and finally, we conclude our work in Section 6.

II. BACKGROUND

In 2008, Zhong et. at [44] have modified the RDF model for representing the heterogeneous data in a dataspace. The distinction between triple model and RDF model with the help of an example shown in Figure 1. Figure 2 exhibits the representation of the data set shown in Figure 1 using RDF model and triple model respectively. From the Figure 2, it is clear that the triple model represents the heterogeneous data without losing their semantics instead of RDF model. Therefore, there is requirement of new set of transformation rules and a set of automatic wrappers. A wrapper is a program which the extract the data from its respective data source and translate them into the desired format. In literature, there are a lot of works have been cited as RDFizer [7, 18, 13, 19] or triplification [11, 34, 35] for transforming the non-RDF data into RDF data which may result the lose of semantics.

The triple data model is a graph based data model which represents both data and relationships among them using triples. A *triple*, represented by symbol τ, is the smallest modeling unit in the triple data model and has three components *S*, *P*, and *O*; where S stands for a *subject* component, P stands for a *predicate* component, and O stands for an *object* component. Formally, we represent a triple as τ = (S, P, O). The subject component *S* is a unique identifier for a data item; it is an integer type and indentifies a data item. The predicate component *P* is a 2-tuple (*l*, *d*), where *l* and *d* refer a label on a type definition of an object component and its data type respectively. Here, *l* and *d* are denoted by finite strings. The object component *O* is an array of bytes and stores the data. Other terms related triple data model are as under:

A *data item* (π) is a unit populated in a dataspace that constitutes data corresponds a real world entity, a relation, a tuple, an xml element, a database, a file/folder, a web page, etc. A data item (π) is decomposed into triples (τ_1^{π}, ..., τ_n^{π}) before populating it in dataspace. The notation τ_i^{π} refers to i^{th} triple of data item (π). Figure 3.1 exhibits an employee data item (π_1) decomposed into triples. Here, employee data item (π_1) is decomposed into a set of triples as {(π_1, (emp_name, string), 'R. Kumar'), (π_1, (date_of_birth, date), '17/11/1983'), (π_1, date_of_joining, date), '15/07/2009'), (π_1, (organization, string), 'NIT'), (π_1, (department, string), 'Computer engineering department'), and (π_1, (salary, currency), Rs 41,543/-)}.

A *data item class* C (π) is a class predefined by a data model for a data item π. The data items share 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 should belong to a predefined data item class otherwise we need to define a new class for the same, e.g., a resource view class for a resource view data item in iDM model [JenM06].

A *triple graph (G)* represents relationships among triples populated in a dataspace. A triple graph (*G*) is defined as *G* = (*N, E, L*), where N is a set of nodes in G that are either internal or leaf nodes. Internal nodes represent data items with their identifications and leaf nodes represent data values. E represents a set of edges in G that are either association edges or attribute edges. An association edge represents a relationship between two data items, and an attribute edge represents a relationship between a data item with its corresponding value for the label on property P. The association and attribute edges are denoted by triples <*dataitem, association_name, dataitem*> and <*dataitem, attribute_name, value*> respectively. Figure 3 exhibits representation of association and attribute edges in a triple graph. *L* is a set of labels on edges representing either an *attribute* or *association* name. Figure 4 illustrates an example of triple graph in which the internal nodes are represented by ovals and leaf nodes are represented by dotted ovals. In a triple graph, an edge is directed from a subject component towards an object component and is labeled by a predicate component of a triple.

Now, we discuss an algorithm based on the decomposition theory of triple model [MiMQ08]. This algorithm maps the modeling constructs of a data model into the triple data model using triples. The algorithm works in two steps: (1) *Step-1* identifies all data item classes belonging to a data model and (2) *Step-2* decomposes each class into its components by applying the function (R^C) and encapsulates each component by a set of triples. Here, R^C is a decomposition function and is defined as $R^{Cj}(\pi_i) = \{r_1^{Cj}(\pi_i) \cup .... \cup r_m^{Cj}(\pi_i)\}$. Algorithm 1 describes the mapping algorithm.

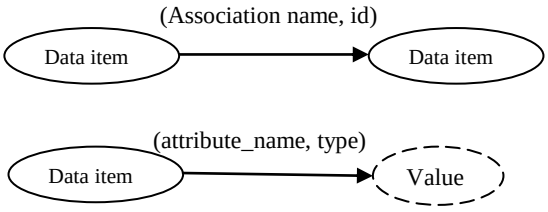


Figure 3 Representation of association and attribute edges in triple graph

Algorithm 1: Decomposition Algorithm

Input: Data Model (D)

Output: Collection of triples $(\tau_1^{\pi}, ..., \tau_n^{\pi})$, n ≥ 1

Step 1:

for each data item $\pi_i \in D$
identify a corresponding data item class $C(\pi_i)$ in *D* to which data item π_i belongs;
if data item class $C(\pi_i)$ does not exist define a new data item class $C(\pi_i)$;

Step 2:

for each data item class $C(\pi_i)$ identified in Step 1
perform decomposition on data item class $C(\pi_i)$ using function R^C to decompose it to set of triples $(\tau_1^{\pi i}, ..., \tau_n^{\pi i})$, where $R^C(\pi_i) = \{r_1^C(\pi_i) \cup .... \cup r_m^C(\pi_i)\}$, $m \geq 1$, and $r_k^C(\pi_i)$ is a decomposition unit or component of the class $C(\pi_i)$, $1 \leq k \leq m$, $n \geq 1$, and the j^{th} triple $\tau_j^{\pi i} = (\pi_i, (a_j, d_j), v_j)$;

end

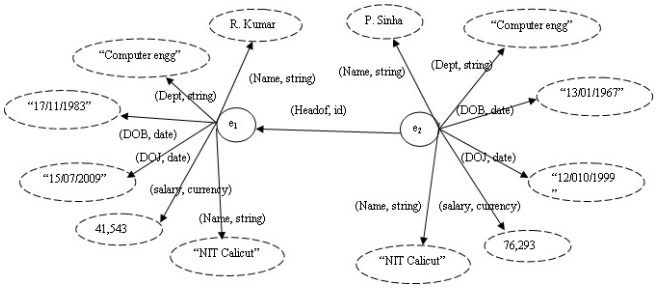


Figure 4 An example of Triple Graph

III. TRANSFORMATION RULE SETS

In this section, we present the set of transformation rules for structured, semi-structured and unstructured data models in summarized way. The detailed discussion on these rules are presented in our previous work [38]. Transformation Rule Sets (TRSs) translate a data model into the triple data model without losing semantics of data. The TRSs depend on the basic properties and the underlying structure of a data model. A TRS may have one or several rules for translating a data item class into triples. The data models that share common properties such as underlying structure, representation of data, etc. belong to either of structured, semi-structure or unstructured data models.

A. 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) [41], [42]. 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., $C^{structured}(\pi)$ = {*database, relation, tuple, attribute*}. Zhong et al. have proposed the rules for decomposing the relation data items and the tuple data items w.r.t. relational model [24]. Now, we propose the rules w.r.t. all the data item classes present in a structured data model. By applying Algorithm 1 on the structured data model, the TRSs will be as follows:

Consider a structured database item (π), named N_{sdb}, consisting of r number of relation items $\{\pi_1,....,\pi_r\}$, where $r \geq 1$. A relation item (π_i), named N^i_{rel}, consisting of s number of tuples $\{\pi_{i1},...., \pi_{is}\}$, where $1 \leq i \leq r$ and $s \geq 0$. A tuple data item (π_{ij}) in relation (π_i) has a set of attributes $(a_{ij1},\ldots,a_{ijn})$, where $1 \leq i \leq$ r, $1 \leq j \leq$ s and $n \geq 1$. An attribute named a_{ijk} has type t_{ijk} and value v_{ijk} for tuple π_{ij} in a relation π_i, $1 \leq i \leq$ r, $1 \leq j \leq$ s, and $1 \leq k \leq$ n. The decomposition of an attribute (a_k) depends on its attribute type.

TRS-1: For C(π) = database

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

TR 1.1: Name Component

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

TR 1.2: Relation Component

$$r_2^{database}(\pi) = \{(\pi, (relation, id), \pi_1), ..., (\pi, (relation, id), \pi_r)\}$$

TRS-2: For C(π_i) = relation

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

TR 2.1: Name Component

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

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: For C (π_{ij}) = tuple

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

TR 3.1: Attribute Component

$$r_1^{tuple}(\pi_{ij}) = \{(\pi_{ij}, (a_{ij1}, t_{ij1}), v_{ij1}),\ldots,(\pi_{ij}, (a_{ijn}, t_{ijn}), v_{ijn})\}$$

TRS-4: For C (a_{ijk}) = attribute

Case 1: For t_{ijk} = b-type

$$R^{attribute}(a_{ijk}) = r_1^{attribute}(a_{ijk})$$

TR 4.1: Attribute Component

$$r_1^{attribute}(a_{ijk}) = (\pi_{ij},(a_{ijk}, t_{ijk}), v_{ijk})$$

Case 2: For t_{ijk} = r-type

We assume that the attribute a_{ijk} has m number of sub-attributes $\{b_{ijk1},...,b_{ijkm}\}$ with data types $\{d_{ijk1},...,d_{ijkm}\}$; sub-attribute b_{ijkl} has the value v_{ijkl} for the k^{th} attribute of j^{th} tuple of i^{th} relation. In this case, the attribute item class decomposes into two components namely name and sub-attribute components.

TR 4.1 Attribute Component

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

TR 4.1.1: Name Component

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

TR 4.1.2: Sub-Attribute Component

$$r_2^{attribute}(a_{ijk}) = \{(\pi_{ij},(b_{ijk1}, d_{ijk1}), v_{ijk1}), ...,(\pi_{ij},(b_{ijkm}, d_{ijkm}), v_{ijkm})\}$$

Case 3: For t_k = s-type

Let us assume that the multi-valued attribute a_{ijk} has a list of m values $\{v_{ijk1},...., v_{ijkm}\}$ with data type d_{ijk}.

$$R^{attribute}(a_{ijk}) = r_1^{attribute}(a_{ijk})$$

TR 4.1: Attribute Component

$$r_1^{attribute}(a_{ijk}) = (\pi_{ij},(a_{ijk}, d_{ijk}[]), \{v_{ijk1,....,} v_{ijkm}\})$$

Case 4: For t_k = ref-type

$$R^{attribute}(a_{ijk}) = r_1^{attribute}(a_{ijk})$$

Let us that the attribute a_{ijk} in one relation refers to an attribute b_{ijk} in another relation.

TR 4.1: Attribute Component

$$r_1^{attribute}(a_{ijk}) = (\pi_{ij},(a_{ijk}, id), b_{ijk})$$

Online Book Store Database (OBSDB) – π

Book (π_1)

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

(a)

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}

(d)

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}

(b)

Publisher (π_3)

Name	Address	Phone	URL	
TMH	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}

(c)

Figure 5 An Example of Structured Database

Now, we demonstrate the decomposition process of a structured database into triples, and construct a triple graph among them with the help of an example. Figure 5 an example of structure database, i.e., online book store database (OBSDB), has 4 relations, and each relation has a number of tuple. By applying the TRS on OBSDB, the partial results of decomposition in triple graph representation is shown in Figures 6.

B. Semi-structured Data Model

A semi-structured data model has different requirements from a relational data model to design schemas to its data. The schema is simple and flexible and may vary for similar objects, i.e., similar objects may have different number of attributes [43]. A unified data model for a semi-structured data is tree/graph-like structures where element sets and attributes are organize as a hierarchical structure [44, 45]. The examples of a semi-structured data are XML data, personal data, and all other data that model as a tree or graph like structure. Therefore, a semi-structured data model has a set of *non-terminal* and *terminal* nodes. A non-terminal node has a label/name, zero or more attributes, and an ordered set of children nodes that can be terminal or non-terminal nodes, i.e., children ϵ {*non_terminal, terminal*}. A terminal node has a label or name, zero or more attributes, and it's content.

Consider a non-terminal node data item (π_i) has a label N_{nt}, n number of attributes $\{ a^{nt}_{i1} ,..., a^{nt}_{in} \}$ with their data types $\{ d^{nt}_{i1} ,..., d^{nt}_{in} \}$ and values $\{ v^{nt}_{i1} ,..., v^{nt}_{in} \}$, $n \geq 0$. The node item (π_i) has m number of children with ids $\{\pi_{i1},..., \pi_{im}\}$, where $m \geq 1$. A terminal node (π_{ij}) label N^t_{ij}, p number of attributes $\{ a^t_{ij1} ,..., a^t_{ijp} \}$ with their data types $\{ d^t_{ij1} ,..., d^t_{ijp} \}$ and values $\{ v^t_{ij1} ,..., v^t_{ijp} \}$ respectively, $n \geq 0$; the content of a terminal node is C_{ijk}. The TRSs are defined under as:

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

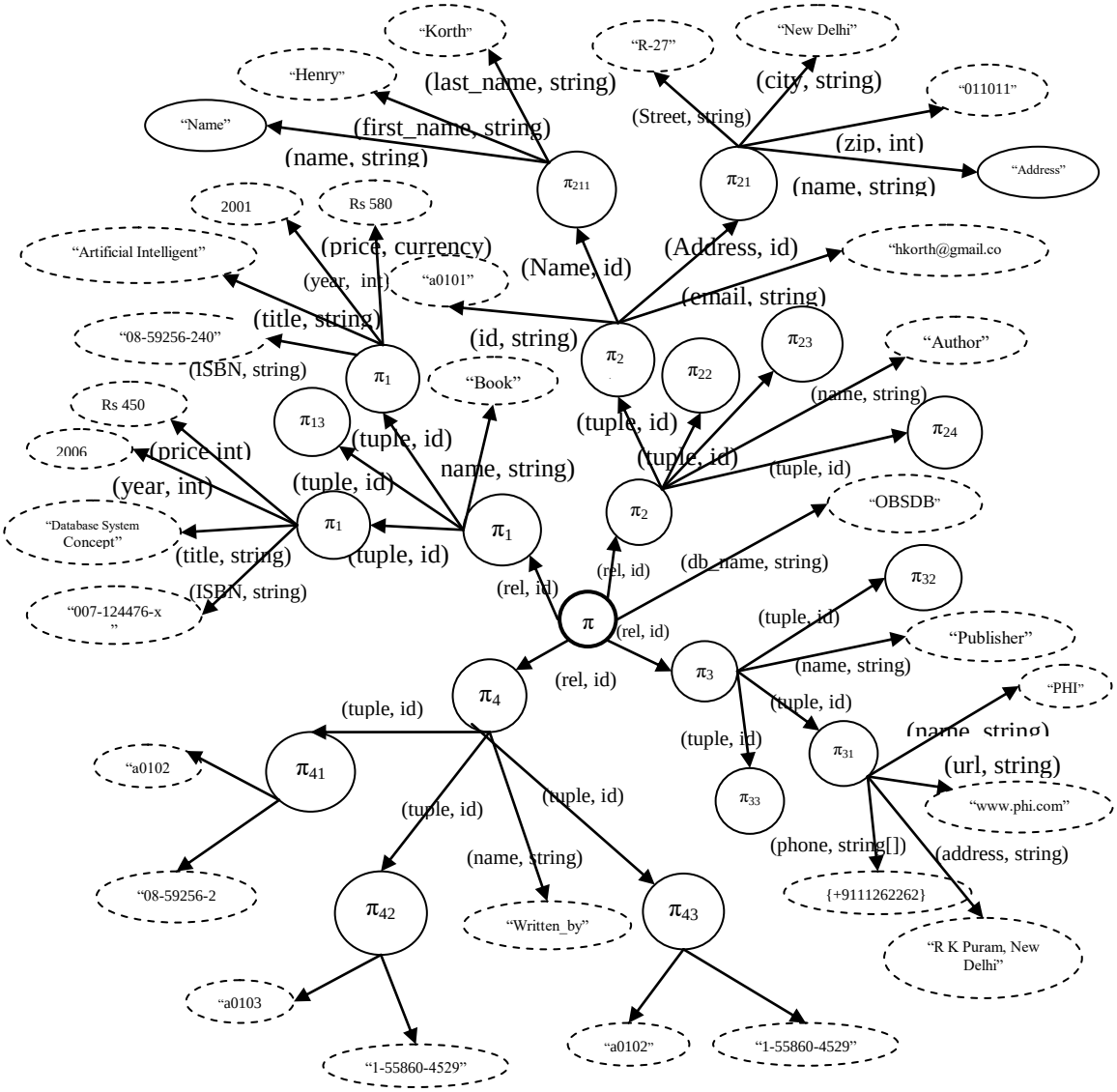


Figure 6 Triple Graph representations of Structured Data

TRS-5: For $C(\pi_i)$ = non-terminal

$R^{non\text{-}terminal}(\pi_i) = r_1^{non\text{-}terminal}(\pi_i) \; U \; r_2^{non\text{-}terminal}(\pi_i) \; U \; r_3^{non\text{-}terminal}(\pi_i)$

TR 5.1: Label Component

$r_1^{non\text{-}terminal}(\pi_i) = (\pi_i, (label, string), N_{nt})$

TR 5.2: Attribute Component

$r_2^{non\text{-}terminal}(\pi_i) = \{(\pi_i, (a_{i1}^{nt}, d_{i1}^{nt}), v_{i1}^{nt}), \ldots, (\pi_i, (a_{in}^{nt}, d_{in}^{nt}), v_{in}^{nt})\}$

TR 5.3: Children Component

$r_3^{non\text{-}terminal}(\pi_i) = \{(\pi_i, (child, id), \pi_{i1}), \ldots, (\pi_i, (child, id), \pi_{im})\}$

TRS 6: if $C(\pi_{ij})$ = terminal

$R^{terminal}(\pi_{ij}) = r_1^{terminal}(\pi_{ij}) \; U \; r_2^{terminal}(\pi_{ij}) \; U \; r_3^{terminal}(\pi_{ij})$

TR 6.1: Label Component

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

TR 6.2: Attribute Component

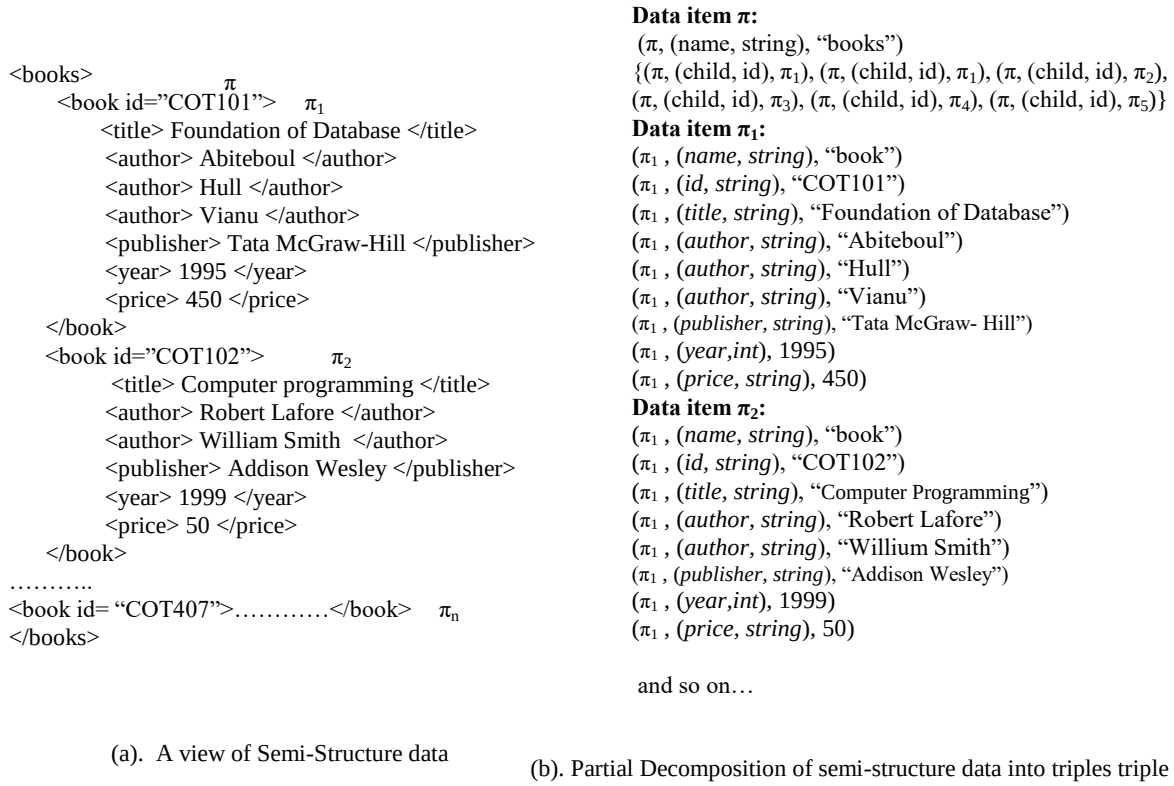
$r_2^{terminal}(\pi_{ij}) = \{(\pi_{ij}, (a_{ij1}^{t}, d_{ij1}^{t}\}, v_{ij1}^{t}), \ldots, (\pi_{ij}, (a_{ijn}^{t}, d_{ijn}^{t}\}, v_{ijn}^{t})\}$

TR 6.3: Content Component

$r_3^{terminal}(\pi_{ij}) = (\pi_{ij}, (content, c\_type), C_{ijk})$

Now, we demonstrate the working of our TRSs by applying them on a semi-structured data. Figure 7(a) exhibits a view of a semi-structured data. By applying the TRS-5 and TRS-6, the results of decomposition into collection of triples are shown in Figure 7(b).

The TRS-5 and TRS-6 can be applied on the existing semi-structured data that has the similar properties as ours such as XML data model, file system data model (i.e., personal data) etc. Consider an XML data model that organizes XML data as an XML tree (i.e., a DOM tree) which consists of mainly XML element and XML text nodes [ToMG05].



(a). A view of Semi-Structure data

(b). Partial Decomposition of semi-structure data into triples triple form

Figure 7 An example of semi-structure data and its triple representation

Corresponding to a semi-structured data model, the XML element and XML text node map with non-terminal and terminal nodes respectively, i.e., *xmlelement* ϵ *non_terminal*, and *xmltext* ϵ *terminal*. Therefore, the data item classes corresponding to an XML data model are *xmlelement* and *xmltext*. Similarly, for personal data, the folder and file node correspond to non-terminal and terminal nodes of a semi-structured data respectively, i.e., *folder* ϵ *non_terminal*, and *file* ϵ *terminal*, therefore, the data item classes corresponding to a personal data are *folder* and *file*.

On the other hand, we can apply the decomposition process on the data models that have distinguished properties from ours such as iDM model [13], Interpreted Object Model (IOM) [27], etc. The iDM model is a graph based data model that represents personal data in dataspace as a resource view graph, where each vertex is a resource view that has four components: name component, tuple component, content component, and group component [JenM06]. Therefore, to design TRSs for iDM model, a resource view data item is mapped into a resource view data item class, i.e., C(π) = {*resourceview*}. Here, a resourceview class decomposes into four components, i.e., name component, tuple component, content component, and group component). Similarly, we can also apply *Algorithm 1* on Interpreted Object Model (IOM) [27]. An IOM is also a personal data model that is used to personal data in a PDSMS. This model represents the personal data as a logical data graph, where each vertex is an Interpreted Object (IO), and each edge represents a relationship between two IOs. the basic modelling unit in the IOM model is an IO that consists of two components: tuple or structured component and content or unstructured component. Therefore, to design the TRSs for IOM model, an IO data item is mapped into an interpreted object class, i.e., C (π) = {*interpretedobject*}. Here, a interpretedobject class decomposes into two components, i.e., tuple component and content component. The tuple and content components have same characteristics as iDM model. The next subsections explain the TRSs for the xml data and personal data by applying the TRS-5 and TRS-6.

C. Unstructured Data Model

Unlike structured and semi-structured data models, an unstructured data model does not have any predefined underlying structure, and represents its data as a sequence of data (or tuple) stream [13]. The examples of an unstructured data includes text, e-mail, web data, multimedia data such as audio, video, image, graphics, etc. An unstructured data model organizes its data as a tree/graph explicitly [46, 47]. We consider the organization of an unstructured as a sequence of data segments, and a data segment contain other data segments and/or data elements. 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 model is decomposed into a sequence of data segments and each data segment is decomposed into a collection of data segments and/or data elements, and each data element is translated into a triple. Consider an example of a data segment "Mr Beans is a member of an organization X"; this data segment decomposed into triples as (*id*, (name, string), "Mr. Beans") and (*id*, (isMemebrof, string), organizationX), where *id* is a unique identifier.

Consider an unstructured document file (π) has name N_{file}, a set of attributes ($a_1,\ldots,a_n$) with data types ($d_1,\ldots, d_n$) and values ($v_1,\ldots, v_n$), and the content of unstructured file is represented as (π_{un}) that are further decomposed into a number of data segments ($\pi_1, \ldots, \pi_m$), where $m \geq 0$. Let a data segment (π_i) has name N_{seg} which is *optional*, and a sequence of p number of children ($\pi_{i1}, \ldots,\pi_{ip}$), where $p \geq 0$ and *children* ϵ {*datasegment*, *dataelement*}. A data element (π_{ij}) has contents which may further decomposed into r units (if required) with values ($v_{ij1},\ldots,v_{ijr}$) and data types ($d_{ij1},\ldots,d_{ijr}$). The TRSs are defined under as:

TRS-7: For C(π) = documentfile

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

TR 7.1: Name Component

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

TR 7.2: Attribute Component

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

TR 7.3: Content Component

$r_3^{documentfile}(\pi) = (\pi, (content, id), \pi_{un})$

TRS-8: For C (π_i) = datasegment

$R^{datasegment}(\pi_i) = r_1^{datasegment}(\pi_i) \; U \; r_2^{datasegmente}(\pi_i)$

TR 8.1: Name Component

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

TR 8.2: Children Component

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

TRS-9: For C (π_{ij}) = dataelement

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

TR 9.1: Content Component

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

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 [31-33] because the existing IE tools have the following disadvantages [48]: 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 [30], 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. [48], 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".

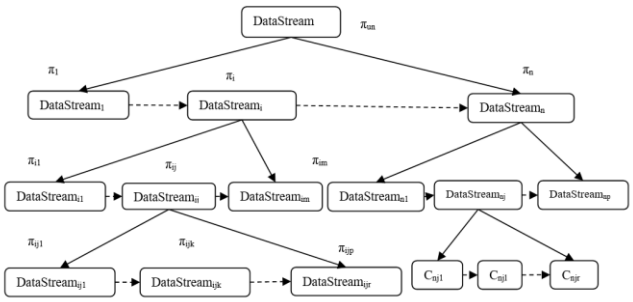


Figure 8 Representation of an unstructured data

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 [48]. 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 [34-36].

IV. RULE-BASED WRAPPER

Figure 9 exhibits the process of transforming the heterogeneous data into triples. The wrapper module consists of a set core wrapper for different type of data sources. This module takes data from the data sources as an input, and transforms them into a collection of triples.

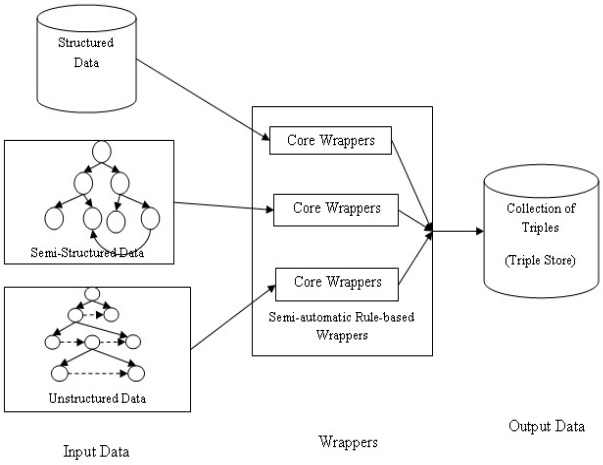


Figure 9: Process of transforming heterogeneous data into triples

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

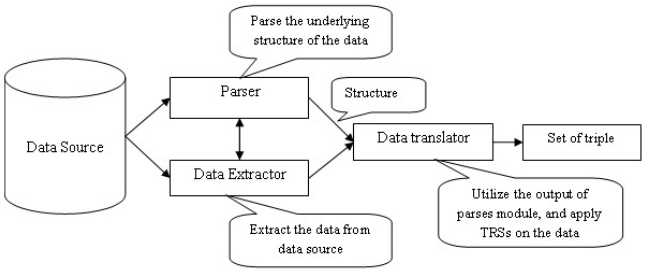


Figure 10: Structure of Core Wrapper module

We have developed the core wrapper in the line of DSP tool architecture [39]. A core wrapper module consists of three sub-modules: parser, data extractor, and data translator. Figure 10 demonstrates the structure of core wrapper module. The parser module analyzes the structure of the underlying data either automatically or manually because the TRSs are mainly dependent on the underlying structure of data in a data source. The data extractor module extracts the desired data from the data source. The data translator module converts the extracted data into set of triples using TRSs. The parser and data extractor modules utilize the functionalities of each other as required. The translator module utilizes output of the parser module, and applies the TRSs on data, and converts them into triple. During our implementation, we have not implemented these modules individually, but the working of our core wrappers implementation is based on these modules.

A. *Structured data*

Designing a core wrapper for structured data is straightforward due to their fixed structure. However, the syntax of the data may differ depending on the database vendors such as mysql, postgress, etc.

We design a core wrapper for the structured data which includes relational as we as object relational data. The core transforms data into a set of triples without explicitly mentioning that whether data is either relational or object relational. In this case, our parser module has utilized the functionality of underlying DBMSs to extract the structure of the data. We have explored the data dictionary of the underlying DBMSs. The data extractor module retrieves the data from the data sources based on the system query. This module retrieves the desired data from data sources in following manners: complete database (all relations) at a time, one or more relations, or a set of tuples from one or more relations. As a result, our wrapper is suitable for all kind of structured data, and extracts the desired information from the various data sources in pay-as-you-go manner. Our translator module is based on the proposed TRSs, which transforms the data as well as relationships into the collection of triples. The wrapper module is called rule-based because it consists of a sub-module based on the proposed TRSs.

We have implemented a single wrapper for structured data based on the TRS-1, TRS-2, TRS-3, and TRS-4. This wrapper is independent from the data sources. In case of structured data, there is no need to develop individual wrappers w.r.t. each data source. The core wrapper needs only a configuration file to configure the wrapper on the data sources w.r.t. their respective database vendors such as mysql, oracle, postgreSQL etc. The configuration file consists of the following information: a connection string w.r.t. database vendor for connecting the data source, data source authentication, and location information. Once we have configured the wrapper on data sources, we can extract the desired information from them by posing the query on the respective data sources, and convert the extracted data into the set of triples.

We have verified our wrapper on the various data sets which was stored on local as well as remote systems. Our data sets are based on mysql-5.1.50, postgresql-8.3.19-1, and Oracle 10g express edition (OracleXE) under Linux (Ubuntu 12.04) and windows 7 operating systems. As a result, the core wrapper for structured data does not depend on underlying operating systems, types of structured data, and underlying database vendors.

B. *Semi-structured data*

Designing a single core wrapper for the semi-structured data is not an easier due to their varying level of heterogeneity at their structure, syntax, and ill-formed contents. On the other hand, the semi-structured data share a common property which is their underlying structure i.e., tree/graph structure, which can be parsed easily. We have designed a set of core wrappers for the semi-structured data, such as XML data, web data, and desktop data etc., w.r.t their underlying structure, and integrate them into a single core wrapper. We have implemented the core wrappers for desktop data (files/folders), semi-structured content of XML and LATEX files, and web data. Our wrappers are independent from the underlying structure of these data.

Table 1: Characteristics of our Data Sets

Characteristics / Data Set	Contents	
DS-1	File system and database	Local Hard drive
DS-2	Web data	Web sites
DS-3	Databases	Remote System
DS-4	E-mails	Email-server

Table 2: Comparison between the proposed work with existing work

Approaches / Kinds of Data	Data	Existing		Proposed	
		TRSs	Wrappers	TRSs	Wrappers
Structured Data	Relational Data	√	√	√	√
	Object Relational Data	X	X	√	√
Semi-Structured Data	Personal Data	√	√	√	√
	XML Data	√	√	√	√
	Web Data	X	X	√	√
	E-mail Data	√	√	√	√
	Latex File	X	X	√	√
	Other Semi-structured Data	X	X	√	√
Unstructured Data	Contents of Files	√	√	√	√
	Multimedia Data	√	√	√	√
	Other unstructured Data	X	X	√	√

We have used the DOM tree parser for the XML and web data, and parsed their structure in breath-first manner. The BFS algorithm is suitable for the XML and desktop data, but it is not an efficient for parsing the web data. It is taking time to for parse them, and identifying the desired data in a web page, because the web page contains much ill-formed contents like html tags. Extracting the data from a huge amount of mix data available in a World Wide Web (WWW) is not straightforward. A web site is a container of mixed data (either semi-structured or unstructured), which contains surface (static) as well as deep or hidden (dynamic) web data. The static contents are HTML or any markup language contents, while the dynamic contents of a web site are changing with time, and come from the deep web [12]. The popular web search engines [14], like Google [6] and other web search engines, return only the static contents of a web sites stored in its crawler database. The dynamic contents are originated from any other sources like hidden web site database, and not indexed by the web crawlers. Therefore, developing an automatic rule-based wrapper is not an easier task. In this case, our parser modules have utilized the functionality of DOM tree parser for analyzing the structure of the XML and web data. The data extractor module extracts the desired data in payas- you-go manner with the help of the parser module. The data extractor modules for web data are needed to identify the data region in the given web page manually. Therefore, this module is semiautomatic.

We have tried to adapt DEPTA algorithm [43] for mining the data records from a web page, but this algorithm is suitable for extracting the single section data records. Our data extractor module for semi-structured data extracts the data in pay-as-you-go manner. For example, the data extractor for xml data takes an XML file or XML file with a set of matching elements or data as an input, retrieve the desired content from the file. The translator modules take the extracted data as an input, apply the respective TRSs, and convert them into set of triples. In our implementation, we have combined all three modules into a single module instead of implementing the individual module, but the working of this module is in same fashion.

This module is called the core wrapper module. The core wrapper module for the web data is semi-automatic and rule based, while the core wrappers for desktop data and XML data are fully automatic. The core wrapper for web data is semi-automatic because they take an URL or URL with a set of keywords as an input, parse the given web page using the HTML DOM tree parser, and required to identified the data region in the give web page manually, and convert them into set of triples. We have applied our web wrappers for extracting the faculty profile records from the various National Institute of Technology (NITs) [1, 2, 3, 4, 5], and found that the wrappers are suitable for extracting the surface web data, which extract the data from the web page in pay-as-you-go manner. Therefore, we have verified that our rules for the semi-structured data like the desktop data, XML data, and surface web data.

C. Unstructured data

In general, the unstructured data consist of multimedia data which do not have a predefine structure. Their structure can be analyzed at run time either manually or semi-automatically, e.g., if a user specify that the specific file is a research article, then it will be easy to predict the approximate structure of the content. Therefore, designing the single core wrapper for unstructured data is a difficult job due to their unspecific structure. We have design a wrapper for extracting the content of a file as a plain text, and encapsulating the whole content into a single triple.
In this case, the role of parser module was played by the human explicitly because of their undefined structure. Our data extractor module extracts the data from the unstructured data sources e.g., a text data files. The core wrappers for unstructured data are semiautomatic.

Once the structure of the data is being defined explicitly, the translator module will translate the data by applying the TRSs. We have implemented a core wrapper for the various unstructured data, and applied our wrapper for extracting the content of the various text file formats, like txt, pdf, tex, bib, rtf, various Microsoft office and open office documents formats etc., located on the local hard drive. We have also defined a set of parser modules explicitly for the bibtex files, research articles etc., and applied the translator module to convert them into the set on triples. The other wrappers extract the content of a file, treated these contents as a string array, and translated them into a single triple. We have verified our wrapper under the Linux and Windows operating system.

We have tried to extract the structured content of the files, but it is a tedious task. The automatically extraction of structured content of a file is unrealistic. We need to examine the structure of a file manually, and implement the wrappers w.r.t. them to extract. This process seems to pretty obvious,
but not suitable for the dataspace system. The transformation process should be as automatic as possible. One possibility is that we manually categorizes the documents based on their content such as research article, book, book chapter etc., analyze their structure, and define a template for a particular category. Otherwise, users will manually specify the various parts (sections, paragraphs etc) of the content structure to model them into triple form. Therefore, designing the automatic core wrappers for unstructured data is not a simple task. We have designed the several semi-automatic rulebased wrappers for the unstructured data. We have categorized the text documents in different categories, designed the set of templates by manually parsing their structure, and the implemented a set of wrappers w.r.t. the design templates.

V. RESULTS AND DISCUSSION

This section presents the evaluation of our work. The goal of this experiment is verifying the proposed TRSs, and evaluating the performance of our wrappers. We have experimentally evaluated our work to show the expressive power of the proposed TRSs, and compare our work w.r.t the existing one. The following sections present the detailed experimental results and discussion about our work.

A. Experimental Setup

We implemented our wrappers using Java and Python language. We used the jdk1.7.0 10 tool kit, and eclipse HELIOS as an IDE for java and Python 2.7 for developing the core wrappers. We used the Ubuntu 12.04 as its operating system with ext4 file system. We required the various java and python libraries for the wrapper development. The wrappers have been verified under the various Windows and Linux based operating system. We have used the real as well synthetic data sets to evaluate the efficiency of our wrappers. Our data sets consist of the several databases (structured ata) on mysql-5.1.50, postgresql-8.3.19-1, and Oracle 10g express edition (OracleXE) platform, XML data, web pages, and personal data under windows and Linux file systems. The personal data consists of mix of heterogeneous data such as directories and files. Our web data sources include the web sites of various NITs [1, 2, 3, 4, 5].

We have extracted the faculty profiles from different NITs with their contact information, research area, publications, and so on. The data sets used in this implementation are taken from the various databases stored in local and remote systems, file systems, and web sites. We prepared 4 types of hypothetical data sets to test our rule based wrappers: Data Set-1 (DS-1), Data Set-2 (DS-2), Data Set-3 (DS-3), and Data Set-4 (DS-4). The characteristics of the data sets are shown in Table 1. We show each data set, their content types, and their location. DS-1 contained a Mysql database, folders, and mix of various types of files, such as text, doc, PDF, Latex, XML,html, and so on, hosted on the local hard drive. In our experiments, we ignored reading the content of the audio/video files because reading the content of these file is unrealistic. DS-2 contained the collection of the selected NITs web site. We created a cluster of web sites of several NITs in DS-2. DS-3 was a cluster of the databases stored on mysql-5.1.50, postgresql-8.3.19-1, and Oracle 10g express edition under the Linux and Windows OS based remote systems. DS-4 consists of the e-mails of one of the author account, which was configured on the Microsoft Office Outlook.

B. Data Extraction and Transformation

For the data extraction, we used all the four data sets in our experiment. We configured the respective wrappers on the data sources, and extracted the desired data from the data sources in pay-as-you-go manner, and converted them into a set of triples. We used the various data extraction approaches for retrieving the data from various data sources (such as structured queries, keyword searching etc.) depending on their content types. We tested our wrappers on the experimental data sets. We configured the core wrappers on the respective data set depending on their contents types, and extracted their contents in triple form by explicitly mentioning the desired data as an input to our wrappers. DS-1 required to configure a core wrapper w.r.t. structured data source (i.e., MySQL database), and a core wrapper w.r.t. the personal data (i.e., file system). Similarly, we configured the core wrappers on DS-2, DS-3, and DS-4 w.r.t. their respective content types, and extracted the data from them in triple form. DS-2 required a set of web wrapper on w.r.t. the similar web pages of different NITs.

Table 3: Comparison with existing work

Wrappers / Kinds of Data	Data	Existing	Proposed
Structured Data	Relational Data	Manual	Automatic
	Object Relational Data	X	Automatic
Semi-Structured Data	Personal Data	Manual	Automatic
	XML Data	Manual	Automatic
	Latex File	X	Automatic
	Web Data	X	Semi-automatic
Unstructured Data	Contents of Files	Manual	Semi-automatic

C. Experimental Summary

In this section, we briefly summarized our experimental results, and compared the proposed work with existing work [44]. Table 2 and Table 3 show the comparison between the proposed works with existing work. In Table 2, X indicates the existence of TRSs and wrappers, while X indicates that TRSs and wrappers are not exist for those data. From Table 2, the proposed TRSs are generic and rational because they are directly applicable to all kind of data, i.e., structured, semi-structured, and unstructured. While, the existing TRSs are specific for a particular kind of data such as relational data, personal data, content of the files etc. The proposed TRSs will help to design and implementing the automatic core wrappers for the dataspace system. We implemented the set of core wrappers based on the proposed TRSs. As shown in Table 3, our most of the core wrappers are automatic, and extract the data from data sources without a human involvement, and few of them are semi-automatic,and require the human involvement during the data extraction and transformation process. While, the existing rule-based wrappers are manual, and need to implement w.r.t. each data source depending on their content and internal structure of the data.

We have implemented a single core wrapper, and this wrapper is applicable to all the respective data sources without explicitly mentioning the structure of the underlying data. In our previous work [40], we implemented the populator module of TripletDS in the line of wrapper module.

VI. Conclusion and Future Work

A dataspace system requires a set of automatic wrappers for extracting the data from its participants. These wrappers will extract the desired data from their respective data sources, and convert them into a collection of triples. In this work, we have proposed the set of transformation rules for structured data, semi-structured data, and unstructured data, and verified them by implementing a set of core wrappers.

We have implemented the core wrappers for structured, semi-structured, and unstructured data. Our wrappers are rule-based, and automatically extract the desired data from the participants in pay-as-you-go manner, and populate them into the dataspace in converted form. A few wrappers are semi-automatic, because they require human involvement for extracting the data from the data sources.

We have experimentally evaluated the expressive power of the proposed TRSs, and concluded that the proposed TRSs as well as wrappers are suitable for the dataspace system. This is our first step for the designing of semi-automatic rule-based wrappers for a dataspace system. Our next step will construct the wrappers as automatic as possible, and implement an automatic parser w.r.t. unstructured data to extract the structured contents of a file without a human involvement.

REFERENCES

1. Available: Nit allahabad http://www.mnnit.ac.in/.
2. Available: Nit hamirpur http://www.nith.ac.in/.
3. Available: Nit jaipur http://www.mnit.ac.in/.
4. Available: Nit jalandhar http://www.nitj.ac.in/.
5. Available: Nit kurukshetra http://www.nitkkr.ac.in/.
6. Google search engine www.google.com.
7. Simile. rdfizers.
8. Resource description framework (rdf) model and syntax specification, February 1999.
9. S. Abiteboul. Querying semi-structured data. Database Theory-ICDT'97, pages 1–18, 1997.
10. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. International journal on digital libraries, 1(1):68–88, 1997.
11. S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: light-weight linked data publication from relational databases. In Proceedings of the 18th international conference on World wide web, pages 621–630. ACM, 2009.
12. M. Bergman. White paper: the deep web: surfacing hidden value. journal of electronic publishing, 7(1), 2001.
13. C. Bizer and A. Seaborne. D2rq-treating non-rdf databases as virtual rdf graphs. In Proceedings of the 3rd international semantic web conference (ISWC2004), page 26, 2004.
14. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine* 1. Computer networks and ISDN systems, 30(1-7):107–117, 1998.
15. P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. Database Theory-ICDT'97, pages 336–350, 1997.
16. P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. ACM SIGMOD Record, 25(2):505–516, 1996.
17. Y. Cai, X. L. Dong, A. Halevy, J. M. Liu, and J. Madhavan. Personal information management with semex. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pages 921–923. ACM, 2005.
18. F. Ciravegna, A. L. Gentile, and Z. Zhang. Lodie: Linked open data for web-scale information extraction. Semantic Web and Information Extraction SWAIE 2012, page 11.
19. S. Das, S. Sundara, and R. Cyganiak. R2rml: Rdb to rdf mapping language, w3c working draft, 24 march 2011, 2011.
20. J. Dittrich and M. Salles. iDM: A unified and versatile data model for personal dataspace management. In Proceedings of the 32nd international conference on Very large data bases, page 378. VLDB Endowment, 2006.
21. I. Elsayed and P. Brezany. Towards large-scale scientific dataspaces for e-science applications. In Database Systems for Advanced Applications, pages 69–80. Springer, 2010.
22. M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. ACM Sigmod Record, 34(4):33, 2005.
23. A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, page 9. ACM, 2006.
24. A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage years. In Proceedings of the 32nd international conference on Very large data bases, pages 9–16. VLDB Endowment, 2006.
25. S. Jeffery, M. Franklin, A. Halevy, and S. R. Jeffery. Soliciting user feedback in a dataspace system. Electrical Engineering and Computer Sciences University of California at Berkeley, 2007.
26. S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 847–860. ACM, 2008.
27. X. Jiang, X. Sun, and H. Zhuge. A resource space model for dataspace. In Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on, pages 33–41. IEEE, 2010.
28. L. Jin, Y. Zhang, and X. Ye. An extensible data model with security support for dataspace management. In High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on, pages 556–563. IEEE, 2008.
29. M. Lenzerini. Data integration: A theoretical perspective. In Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 233–246. ACM, 2002.
30. Y. Li and X. Meng. Research on personal dataspace management. In Proceedings of the 2nd SIGMOD PhD workshop on Innovative database research, pages 7–12. ACM, 2008.
31. Z. Liu, Y. Mao, J. Wang, and C. Ye. A data model for web Dataspaces management. In Education Technology and Computer Science, 2009.ETCS'09. First International Workshop on, volume 2, pages 137–141, IEEE, 2009.
32. H. T. Mirza, L. Chen, and G. Chen. Practicability of dataspace systems. JDCTA, 4(3):233–243, 2010.
33. A. Ndjafa, H. Kosch, D. Coquil, and L. Brunie. Towards a model for multimedia dataspaces. In Multimedia on the Web (MMWeb), 2011 Workshop on, pages 33–37. IEEE, 2011.
34. A. G. Nuzzolese, A. Gangemi, V. Presutti, and P. Ciancarini. Semion: a smart triplification tool. Proceedings of the EKAW2010 Poster and Demo Track, 674.
35. A. G. Nuzzolese, A. Gangemi, V. Presutti, and P. Ciancarini. Finetuning triplification with semion. In EKAW workshop on Knowledge Injection into and Extraction from Linked Data (KIELD2010), pages 2–14, 2010.
36. Y. Pan, Y. Tang, and S. Li. Web services discovery in a pay-as-you-go fashion. Journal of Universal Computer Science, 17(14):2029–2047, 2011.
37. S. Pradhan. Towards a novel desktop search technique. In Database and Expert Systems Applications, pages 192–201. Springer, 2007.
38. M. Singh and S. Jain. Transformation rules for decomposing heterogeneous data into triples. Journal of King Saud University-Computer and Information Sciences, 27(2):181–192, 2015.
39. M. Singh, S. Jain, and V. Panchal. An architecture of dsp tool for publishing the heterogeneous data in dataspace. In Information Technology (ICIT), 2014 International Conference on, pages 209–214. IEEE, 2014.
40. M. Singh and S. K. Jain. Tripletds: a prototype of dataspace system based on triple data model. International Journal of Computational Systems Engineering, 3(3):144–156, 2017.
41. M. Vaz Salles, J. Dittrich, S. Karakashian, O. Girard, and L. Blunschi. iTrails: pay-as-you-go information integration in dataspaces. In Proceedings of the 33rd international conference on Very large data bases, pages 663–674. VLDB Endowment, 2007.
42. D. Yang, D. Shen, T. Nie, G. Yu, and Y. Kou. Layered graph data model for data management of dataspace support platform. Web-Age Information Management, pages 353–365, 2011.
43. Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In Proceedings of the 14th international conference on World Wide Web, pages 76–85. ACM, 2005.
44. M. Zhong, M. Liu, and Q. Chen. Modeling heterogeneous data in dataspace. In IEEE International Conference on Information Reuse and Integration, 2008. IRI 2008, pages 404–409, 2008.
45. M. Zhong, M. Liu, and Y. He. 3sepias: A semi-structured search engine for personal information in dataspace system. Information Sciences, pages 31–50, 2012.
46. Niranjan Lal, Shamimul Qamar, Savita Shivani, "Search Ranking for Heterogeneous Data over Dataspace", Indian Journal of Science and Technology (Scopus Indexed). Volume 9, Issue 36, pp.1-9, 2016
47. Niranjan Lal, Bhagyashree Pathak,"Information Retrieval from Heterogeneous Data Sets using Moderated IDF-Cosine Similarity in Vector Space Model" "International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS 2017)", IEEE Explorer , 2018.
48. F. Kastrati, X. Li, C. Quix, and M. Khelghati, "Enabling structured queries over unstructured documents," in Mobile Data Management (MDM), 2011 12th IEEE International Conference on, vol. 2. IEEE, 2011, pp. 80–85.