Rule-Based Wrappers for a Dataspase 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 dataspase technology addresses this problem in an efficient manner. A dataspase 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 dataspase in desired format (e.g., triple format).

This work presents a set of rule-based wrappers for a dataspase 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 dataspase 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: Dataspase System; Triple Model; Rule-based Wrappers; TripledS; Pay-as-you-go

1. INTRODUCTION

In recent past, the attention has been drawn on effectively managing of distributed and heterogeneous data [29, 24], and excellently solved by the dataspase system [22]. With respect to the other existing data integration systems, a dataspase 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 dataspase system has the distinguish properties from these systems [32]. Therefore, the invention of a dataspase system requires a new data model and a new query language for efficiently representing and retrieving the heterogeneous the data in dataspase. There are several model has been cited in literature [20, 37, 44, 42, 28, 45, 31, 36, 33, 27], and adopted by the various dataspase 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.

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 (i,d), where label i 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 dataspase 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 dataspase systems by extracting the data from the data sources in "pay-as-you-go" manner.

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

Published By:
Blue Eyes Intelligence Engineering & Sciences Publication
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 the paper is organized as follows: Section 2 presents basis of triple model. Section 3 discusses 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. al [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 identifies 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 (τ₁, ..., τₙ) before populating it in dataspace. The notation τᵢ refers to the iᵗʰ triple of data item (π). Figure 3.1 exhibits an employee data item (π₁) decomposed into triples. Here, employee data item (π₁) is decomposed into a set of triples as {((π₁, (emp_name, string), 'R. Kumar'), (π₁, (date_of_birth, date), '17/11/1983'), (π₁, date_of_joining, date), '15/07/2009'), (π₁, (organization, string), 'NIT'), (π₁, (department, string), 'Computer engineering department'), and (π₁, (salary, currency), Rs 41,543/)}.
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 identifiers 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 \(<\text{dataitem}, \text{association}\_\text{name}, \text{dataitem}>\) and \(<\text{dataitem}, \text{attribute}\_\text{name}, \text{value}>\) 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 \( R^c(\pi) = \{\tau^c_1(\pi), \ldots, \tau^c_n(\pi)\} \). Algorithm 1 describes the mapping algorithm.

**Algorithm 1: Decomposition Algorithm**

**Input:** Data Model (D)

**Output:** Collection of triples \( (\tau^e_1, \ldots, \tau^e_n) \), \( n \geq 1 \)

**Step 1:**

for each data item \( \pi \in D 

identify a corresponding data item class \( C(\pi) \) in \( D \) to which data item \( \pi \) belongs;

if data item class \( C(\pi) \) does not exist define a new data item class \( C(\pi) \);

**Step 2:**

for each data item class \( C(\pi) \) identified in Step 1

perform decomposition on data item class \( C(\pi) \) using function \( R^c \) to decompose it to set of triples \( (\tau^e_1, \ldots, \tau^e_n) \), where \( R^c(\pi) = \{\tau^c_1(\pi), \ldots, \tau^c_m(\pi)\}, m \geq 1 \), and \( \tau^c_i(\pi) \) is a decomposition unit or component of the class \( C(\pi), I \leq k \leq m, n \geq 1 \), and the \( j \)th triple \( \tau^e_j = (\pi, (a_j, d_j), v_j) \);

**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., \( C^{\text{structured}}(\pi) = \{\text{database, relation, tuple, attribute}\} \). 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 \((\pi)\), named \(N_{\pi}\), consisting of \(r\) relation of number of attributes \(\{\pi_1, ..., \pi_r\}\), where \(r \geq 1\). A relation item \((\pi_i)\), named \(N_{\pi_i}\), consisting of \(s\) number of tuples \(\{t_{ij1}, ..., t_{ijm}\}\), where \(1 \leq i \leq r\) and \(s \geq 0\). A tuple data item \((\pi_{ij})\) in relation \((\pi_i)\) has a set of attributes \(\{d_{ijk}, ..., d_{ijm}\}\), where \(1 \leq k \leq r\), \(1 \leq j \leq s\) and \(n \geq 1\). An attribute named \(d_{ijk}\) has type \(t_{ijk}\) and value \(v_{ijk}\) for tuple \(\pi_{ij}\) in a relation \((\pi_i)\), \(1 \leq i \leq r\), \(1 \leq j \leq s\), and \(1 \leq k \leq n\). The decomposition of an attribute \((\alpha_k)\) depends on its attribute type.

**TRS-1:** For \(C(\pi) = \text{database}\)

\[ R_{\text{database}}(\pi) = r_1\text{database}(\pi) \cup r_2\text{database}(\pi) \]

**TR 1.1:** Name Component

\[ r_1\text{database}(\pi) = (\pi, (\text{db name}, \text{string}), N_{\pi}) \]

**TR 1.2:** Relation Component

\[ r_2\text{database}(\pi) = \{ (\pi, (\text{relation}, \text{id}), \pi_1), ..., (\pi, (\text{relation}, \text{id}), \pi_r) \} \]

**TRS-2:** For \(C(\pi) = \text{relation}\)

\[ R_{\text{relation}}(\pi) = r_1\text{relation}(\pi) \cup r_2\text{relation}(\pi) \]

**TR 2.1:** Name Component

\[ r_1\text{relation}(\pi) = (\pi, (\text{rel name}, \text{string}), N_{\pi}) \]

**TR 2.2:** Tuple Component

\[ r_2\text{relation}(\pi) = \{ (\pi_1, (\text{tuple}, \text{id}), \pi_1), ..., (\pi_r, (\text{tuple}, \text{id}), \pi_r) \} \]

**TRS-3:** For \(C(\pi_i) = \text{tuple}\)

\[ R_{\text{tuple}}(\pi_i) = r_1\text{tuple}(\pi_i) \]

**TR 3.1:** Attribute Component

\[ r_1\text{tuple}(\pi_i) = \{ (\pi_{ij}, (a_{ijk}, t_{ijk}), v_{ijk}), ..., (\pi_{ij}, (a_{ijn}, t_{ijn}), v_{ijn}) \} \]

**TRS-4:** For \(C(\alpha_k) = \text{attribute}\)

**Case 1:** For \(t_{ijk} = \text{b-type}\)

\[ R_{\text{attribute}}(\alpha_k) = r_1\text{attribute}(\alpha_k) \]

**TR 4.1:** Attribute Component

\[ r_1\text{attribute}(\alpha_k) = (\pi_{ij}, (a_{ijk}, t_{ijk}), v_{ijk}) \]

**Case 2:** For \(t_{ijk} = \text{r-type}\)

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

**TR 4.1.1:** Name Component

\[ r_1\text{attribute}(\alpha_k) = (\pi_{ij}, (\text{name}, \text{string}), a_{ijk}) \]

**TR 4.1.2:** Sub-Attribute Component

\[ r_2\text{attribute}(\alpha_k) = \{ (\pi_{ij}, (b_{ijk}, d_{ijk}), v_{ijk}), ..., (\pi_{ij}, (b_{ijm}, d_{ijm}), v_{ijm}) \} \]

**Case 3:** For \(t_{ijk} = \text{s-type}\)

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

\[ R_{\text{attribute}}(\alpha_k) = r_1\text{attribute}(\alpha_k) \]

**TR 4.1:** Attribute Component

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

**Case 4:** For \(t_{ijk} = \text{ref-type}\)

\[ R_{\text{attribute}}(\alpha_k) = r_1\text{attribute}(\alpha_k) \]

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

**TR 4.1:** Attribute Component

\[ r_1\text{attribute}(\alpha_k) = (\pi_{ij}, (a_{ijk}, \text{id}), b_{ijk}) \]

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 tuples. 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 \(\epsilon\) \{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 \((\pi)\) has a label \(N_{\pi}\), \(n\) number of attributes \(\{a_{11}, ..., a_{1n}\}\) with their data types \(\{d_{11}, ..., d_{1n}\}\) and values \(\{v_{11}, ..., v_{1n}\}\), \(n \geq 0\). The node item \((\pi)\) has \(m\) number of children with ids \(\{\pi_1, ..., \pi_m\}\), where \(m \geq 1\). A terminal node \((\pi)\) label \(N_{\pi}\), \(p\) number of attributes \(\{a_{ij1}, ..., a_{ijp}\}\) with their data types \(\{d_{ij1}, ..., d_{ijp}\}\) and values \(\{v_{ij1}, ..., v_{ijp}\}\) respectively, \(n \geq 0\); the content of a terminal node is \(C_{ij}\). The TRSs are defined under as:

---

*Figure 5 An Example of Structured Database*
Figure 6 Triple Graph representations of Structured Data

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

$$R_{\text{non-terminal}}(\pi_i) = r_{1_{\text{non-terminal}}}(\pi_i) U r_{2_{\text{non-terminal}}}(\pi_i) U r_{3_{\text{non-terminal}}}(\pi_i)$$

TR 5.1: Label Component

$$r_{1_{\text{non-terminal}}}(\pi_i) = (\pi_i, (\text{label}, \text{string}), N_{a_i})$$

TR 5.2: Attribute Component

$$r_{2_{\text{non-terminal}}}(\pi_i) = \{ (\pi_i, (a_{1}^{m_{1}}, d_{1}^{m_{1}}, v_{1}^{m_{1}}), ... ) \}$$

TR 5.3: Children Component

$$r_{3_{\text{non-terminal}}}(\pi_i) = \{ (\pi_i, (\text{child}, \text{id}), N_{i_i}) \}$$

TRS 6: if $C(\pi_i) = \text{terminal}$

$$R_{\text{terminal}}(\pi_i) = r_{1_{\text{terminal}}}(\pi_i) U r_{2_{\text{terminal}}}(\pi_i) U r_{3_{\text{terminal}}}(\pi_i)$$

TR 6.1: Label Component

$$r_{1_{\text{terminal}}}(\pi_i) = (\pi_i, (\text{label}, \text{string}), N_{i_i}^j)$$

TR 6.2: Attribute Component

$$r_{2_{\text{terminal}}}(\pi_i) = \{ (\pi_i, (a_{i}^{j_1}, d_{i}^{j_1}, v_{i}^{j_1}), ... ) \}$$

TR 6.3: Content Component

$$r_{3_{\text{terminal}}}(\pi_i) = (\pi_i, (\text{content}, c_{\text{type}}), C_{i_i})$$

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].
Corresponding to a semi-structured data model, the XML element and XML text node map with non-terminal and terminal nodes respectively, i.e., xml element ∈ non_terminal, and xml text ∈ terminal. Therefore, the data item classes corresponding to an XML data model are xml element and xml text. 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 ∈ non_terminal, and file ∈ 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) = {interpretedobject}. Here, an interpreted object 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, a per unit cost, and an order date are data elements.

(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

Data item π:

(π, (name, string), “books”)

(π, (child, id, π2), (π, (child, id, π3), (π, (child, id, π2), (π, (child, id, π3), (π, (child, id, π2), (π, (child, id, π3), (π, (child, id, π2), (π, (child, id, π3)))

Data item π2:

(π2, (name, string), “book”)

(π2, (id, string), “COT101”)

(π2, (title, string), “Foundation of Database”)

(π2, (author, string), “Abiteboul”)

(π2, (author, string), “Hull”)

(π2, (author, string), “Vianu”)


(π2, (year, int), 1995)

(π2, (price, string), 450)

Data item π3:

(π3, (name, string), “book”)

(π3, (id, string), “COT102”)

(π3, (title, string), “Computer Programming”)

(π3, (author, string), “Robert Lafoare”)

(π3, (author, string), “William Smith”)

(π3, (publisher, string), “Addison Wesley”)

(π3, (year, int), 1999)

(π3, (price, string), 50)

and so on…

(π, (name, string), “books”)

(π, (child, id, π2), (π, (child, id, π3), (π, (child, id, π2), (π, (child, id, π3), (π, (child, id, π2), (π, (child, id, π3), (π, (child, id, π2), (π, (child, id, π3)))

Data item π2:

(π2, (name, string), “book”)

(π2, (id, string), “COT101”)

(π2, (title, string), “Foundation of Database”)

(π2, (author, string), “Abiteboul”)

(π2, (author, string), “Hull”)

(π2, (author, string), “Vianu”)


(π2, (year, int), 1995)

(π2, (price, string), 450)

Data item π3:

(π3, (name, string), “book”)

(π3, (id, string), “COT102”)

(π3, (title, string), “Computer Programming”)

(π3, (author, string), “Robert Lafoare”)

(π3, (author, string), “William Smith”)

(π3, (publisher, string), “Addison Wesley”)

(π3, (year, int), 1999)

(π3, (price, string), 50)
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, (isMemberof, string), organizationX), where id is a unique identifier.

Consider an unstructured document file (π) has name N_file, a set of attributes (a_1, ..., a_n) with data types (d_1, ..., d_n) and values (v_1, ..., v_n), and the content of unstructured file is represented as (π_m) that are further decomposed into a number of data segments (π_1, ..., π_m), where m ≥ 0. Let a data segment (π) has name N_m which is optional, and a sequence of p number of children (π_1, ..., π_p), where p ≥ 0 and children ∈ {datasetsegment, dataelement}. A data element (π_e) has contents which may further decomposed into r units (if required) with values (v_1, ..., v_r) and data types (d_1, ..., d_r). The TRSs are defined under as:

**TRS-7:** For \( C(\pi) = \text{documentfile} \)

\[
R_{\text{documentfile}}(\pi) = R_{\text{documentfile}}(\pi) \cup R_{\text{documentfile}}(\pi) \cup R_{\text{documentfile}}(\pi)
\]

TR 7.1: Name Component

\[
r_{1}^{\text{documentfile}}(\pi) = (\pi, \text{name}, \text{string}), N_{\text{file}}
\]

TR 7.2: Attribute Component

\[
r_{2}^{\text{documentfile}}(\pi) = \{(\pi, (a_1, d_1), v_1), ..., (\pi, (a_n, d_n), v_n)\}
\]

TR 7.3: Content Component

\[
r_{3}^{\text{documentfile}}(\pi) = \{(\pi, \text{content}, \text{id}), \pi_m\}
\]

**TRS-8:** For \( C(\pi) = \text{dataset} \)

\[
R_{\text{dataset}}(\pi) = R_{\text{dataset}}(\pi) \cup R_{\text{dataset}}(\pi) \cup R_{\text{dataset}}(\pi)
\]

TR 8.1: Name Component

\[
r_{1}^{\text{dataset}}(\pi) = (\pi, \text{name}, \text{string}), N_{\text{file}}
\]

TR 8.2: Children Component

\[
r_{2}^{\text{dataset}}(\pi) = \{(\pi, \text{child}, \text{id}), \pi_m\}
\]

**TRS-9:** For \( C(\pi) = \text{dataelement} \)

\[
R_{\text{dataelement}}(\pi) = R_{\text{dataelement}}(\pi)
\]

TR 9.1: Content Component

\[
r_{1}^{\text{dataelement}}(\pi) = \{(\pi, \text{content}, \text{d}_1), v_1(\pi), ..., (\pi, \text{content}, \text{d}_r), v_r(\pi)\}
\]

The proposed TRSs for unstructured data are simple and straightforward. 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-date 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”.

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/video 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.
Rule-Based Wrappers for a Dataspace System

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, postgresSQL 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>
<thead>
<tr>
<th>Characteristics</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Set</strong></td>
<td></td>
</tr>
<tr>
<td>DS-1</td>
<td>File system and database</td>
</tr>
<tr>
<td>DS-2</td>
<td>Web data</td>
</tr>
<tr>
<td>DS-3</td>
<td>Databases</td>
</tr>
<tr>
<td>DS-4</td>
<td>E-mails</td>
</tr>
</tbody>
</table>

Table 2: Comparison between the proposed work with existing work
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 pay-as-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 semi-automatic.

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 semi-automatic.

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 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.
Rule-Based Wrappers for a Dataspace System

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 data) 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

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, 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

7. Simile. rdfizers.