Transforming Data from DataPile Structure into RDF

Jiří Dokulil

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
dokulil@gmail.com

Abstract. Huge amount of interesting data has been gathered in the DataPile structure since its creation. This data could be used in the development of RDF databases. When limited to basic information stored in the DataPile the transformation into RDF is straightforward. It still provides millions of RDF triples with complex structure and many irregularities.

1 Introduction

While it is easy to find huge relational or XML data rich in structure there is still not much data available in the RDF format. Such data could be obtained by simple conversion from a relational database but this data would be simple and with a regular structure.

In this paper we propose a transformation of data stored in the DataPile structure [1] into the RDF [2]. We expect to receive huge amount of RDF triples with more interesting structure and much less regular than data from relational databases. This expectation is based on the way the DataPile structured is being used in practice.

The DataPile system was developed to integrate data from a heterogeneous set of databases. Among main design goals were storage of historical versions of data and easy adaptation to global schema changes.

First of all we present the DataPile and RDF models, then describe the transformation of metadata and data and finally give results of an experimental implementation of the transformation.

2 The Data Models

In this section we present the data models that take part in the transformation.

2.1 The DataPile

The terminology used in DataPile systems is different from those used in relational and RDF databases. *Entity* is a rough equivalent of a table scheme. It has a name and consists of *attributes*, which can be compared to column definitions. Each attribute defines an attribute name and data type. The set of allowed data types had to be very

V. Snášel, K. Richta, J. Pokorný (Eds.): Dateso 2006, pp. 54–62, ISBN 80-248-1025-5.

limited because of implementation reasons. The only supported types are string, number, timestamp, BLOB (Binary Large Object) and a typed reference (foreign key). The DataPile system allows definition of multiple entities each having zero or more attributes. One attribute can not be a member of multiple entities. On the other hand, multiple attributes with the same name can exist, as long as they are members of different entities.

Entities and attributes are *metadata*. They define structure of the actual data that can be stored in the system.

The data consist of *attribute values*. An attribute value is one data item together with type information (identifier of an entity attribute), validity period, relevance and source of the value. Attribute values describing one object are grouped together into an *entity instance*. Each entity instance is assigned a unique eighteen-digit number called *entity instance* identifier. Only attributes of one entity can be used as types of attribute values forming one entity instance. This entity is called *a type of* that *entity instance*.

All entity instances of one type can be viewed as a relational table with each row containing one entity instance. Then one attribute value would be one item of this table.

All this information (matadata and data) is stored in a relational database with special schema called the DataPile structure. This structure and a set of applications and tools form the DataPile system.

In order to achieve the goals set for the DataPile the data could not be stored using one table for each entity type. Instead, a special DataPile structure was created. The center of the structure is one table called PILE capable of storing all attributes of all entities along with their history was used. This table is supplemented by other tables that store metadata, e.g. list of entities and their attributes.

One row of the PILE table contains entity instance identifier, attribute identifier, attribute value, validity period, and other information used in the data integration process. The attribute value is stored in more table columns. It requires one column for each data type. This is the reason why only fixed and very limited set of data types was allowed in the DataPile structure.

Let us look at an example. Consider a system for storing basic information about people. Relational schema could look like this:

```
PERSON(id, first_name, last_name, date_of_birth).
```

In a data pile system this schema would require metadata containing one entity called “PERSON” consisting of three attributes (first_name, last_name and date_of_birth). Data type of first_name and last_last name would be a string in both models. On the other hand there is no exact equivalent for “date” data type, which would probably be the type of the date_of_birth column in a relational database. The “timestamp” data type would have to be used.

Table 1. Example data to be stored in the DataPile

id	First_name	last_name	date_of_birth
1	John	Smith	5.8.1962
2	Jane	Doe	23.2.1971

We can now transform relational data from the Table 1 into the DataPile structure. First of all, both records have to be assigned an entity instance identifier. Normally it would have been an eighteen-digit number but for convenience we use 101 and 102 as the identifiers.

Two instances of entity "PERSON" with identifier 101 and 102 have to be created. Then the appropriate attribute values are to be created in the PILE table. PILE table containing these attributes is displayed in the Table 2.

The table also contains an example of storing historical version of data. On 5.7.2005 the name of Jane Doe was changed to Joan Doe.

Table 2. PILE table with example data (simplified, some columns omitted). Ent_id stands for entity instance identifier.

ent_id	attribute	string value	time value	valid from	valid to
101	first_name	John	*null*	28.5.2005 15:31:20	*null*
101	last_name	Smith	*null*	28.5.2005 15:31:20	*null*
101	date_of_birth	*null*	5.8.1962 0:00:00	28.5.2005 15:31:20	*null*
102	first_name	Jane	*null*	27.5.2005 10:12:25	5.7.2005 9:25:05
102	first_name	Joan	*null*	5.7.2005 9:25:05	*null*
102	last_name	Doe	*null*	27.5.2005 10:12:25	*null*
102	date_of_birth	*null*	23.2.1971 0:00:00	27.5.2005 10:12:25	*null*

2.2 The RDF

One of the goals of the RDF is integration of data gathered about resources on the World Wide Web. Such data tend to be rich in structure and often incomplete.

The RDF is used to make statements about resources. A RDF statement is a triple consisting of a subject, a predicate and an object. This states that the subject has a property (predicate) with a certain value (object). The statement is modeled as a graph with one node for the subject, one node for the object and an arc for the predicate, directed from the subject node to the object node.

A typical example looks like this:

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/>
"John Smith"
```

This states that the book identified by URI <http://example.org/book/book1> was created by John Smith. The book is the subject, "created" is the predicate and John Smith is the object of the triple. In this example we represent John Smith by a literal

"John Smith". A literal is a constant expression that can be typed or untyped (plain). They are used to represent values like numbers and dates by their lexical representation. It is always possible to use URI instead of a literal, e.g. <http://www.example.org/staffid/8574>. Then we could also make statements about John Smith. Literals can only be used as objects while URI can take any place in a triple.

URIs are represented by named nodes in the RDF graph. However we do not always need direct access to every node in the graph. Some nodes are always accessed using arcs from other nodes. These nodes do not need universal identifiers like URIs. They can be created as *blank nodes*. These nodes can be used as subjects and objects. Blank nodes are usually assigned a unique identifier when the graph is serialized to a triples representation. Common way of writing such identifiers is _:*identifier*, e.g. "_:blank123". This identifier represents the same blank node in the whole representation of the graph. Different identifiers represent different blank nodes.

3 The Transformation

The basic idea behind the transformation is that by making a projection of the PILE table on the columns containing entity instance identifier, attribute and attribute value we receive a set of triples representing statements very similar to RDF statements.

3.1 The Entity Instance Identifiers

All entity instances in the DataPile are assigned a unique eighteen digit number called entity instance identifier.

We need a way to create nodes with unique names in the RDF graph that will represent the objects we want to make statements about. The entity instance identifier is ideal for this. It can be used either as a part of an URI represented by the node or as an identifier of a blank node if we choose not to give a name to the node. In this paper we describe the latter approach since we wanted to create data that would help in the development of RDF databases and queries containing or returning blank nodes are an important feature of the database we want to test.

If naming of the nodes is required then the transformation process can easily be modified to create nodes with URIs.

3.2 The Metadata

Processing of the data in the DataPile is controlled by metadata that is stored in relational tables. In order for the transformation to work at least some part of the metadata must be stored in the RDF as well.

The most important piece of metadata to transform is attributes of the entities. They serve as predicates (arcs of the RDF graph). The very basic RDF representation of a single attribute looks like this (TURTLE notation [3]).

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix mt : <http://example.org/stoh/metadata/> .

mt:person__name rdf:type rdf:Property .
```

This defines http://example.org/stoh/metadata/person__name to be an attribute. The original version in the DataPile was an attribute called "name" belonging to an entity called "person". We can represent this information in the RDF as well.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix mt : <http://example.org/stoh/metadata/> .

mt:person rdf:type rdfs:Class .
mt:person__name rdf:type rdf:Property .
mt:person__name rdfs:domain mt:person .
```

Alternatively we could name the attribute only by its name in the DataPile and omit the name of the entity. This would allow us to make queries like "Give me the name of all entity instances that have a name". On the other hand it would complicate type checking of the values. Because of this we chose the more specific names.

With the information about entities we can specify a type (entity) of an entity instance.

```
_:568421369754123695 rdf:type mt:person .
```

The subject of the triple is a blank node with an eighteen digit identifier identical to the entity instance identifier in the DataPile.

3.3 The Data Types

The DataPile uses a limited number of data types for the attributes. They are listed in Table 1 together with their equivalents after the transformation.

Table 3. Data types in the DataPile and after the transformation

string	http://www.w3.org/2001/XMLSchema#string
number	http://www.w3.org/2001/XMLSchema#decimal
timestamp	http://www.w3.org/2001/XMLSchema#dateTime
entity reference	*reference to a blank node*

Using these data types we can extend the transformed metadata representation.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

```
@prefix mt : <http://example.org/stoh/metadata/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

mt:person rdf:type rdfs:Class .
mt:person__name rdf:type rdf:Property .
mt:person__name rdfs:domain mt:person .
mt:person__name rdfs:range xsd:string .
```

The entity references in the DataPile are typed references. One attribute can only be used to reference one specified entity. This is equivalent to specifying one class as a range of a property.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix mt : <http://example.org/stoh/metadata/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

mt:person rdf:type rdfs:Class .
mt:address rdf:type rdfs:Class .
mt:person__address rdf:type rdf:Property .
mt:person__address rdfs:domain mt:person .
mt:person__address rdfs:range mt:address .
```

3.4 Transforming the Data

After storing the necessary metadata in the RDF graph we can start transforming the real data. Since every row of the PILE table contains entity instance identifier, attribute identifier and typed value we can make a simple projection of the PILE table on these columns and create one RDF triple from each row.

An example output could look like this:

```
@prefix xsd : <http://www.w3.org/2001/XMLSchema> .

_:568421369754123695 mt:person__name “John Smith” .
_:568421369754123695 mt:person__date_of_birth
     “1980-08-14T00:00:00”^^xsd:dateTime .
_:568421369754123695 mt:person__height
     “1.82”^^xsd:decimal .
_:568421369754123695 mt:person__father
     _:684258941535789524 .
```

The last triple is a reference to another entity instance (a foreign key).

3.5 Multilingual Attributes

Although the presented general transformation is capable of handling all types of date stored in the DataPile there is one case that could be handled in a better way. Practical application of the DataPile showed that it is sometimes necessary to handle string values that need to be expressed in different languages. For instance name of a department in Czech and English or same word in different cases.

Using the DataPile it was necessary to create two new entities causing this feature to be hard to use. The RDF offers an easier way of achieving the same results. The standard offers a way to specify a language tag for every string literal. The language tags are defined by RFC 3066 [4] which is flexible enough to specify not only language but different cases as well.

```
_:469751359754692454 rdf:type mt:department .
_:469751359754692454 mt:department__name
     “Katedra softwarového inženýrství”@cs .
_:469751359754692454 mt:department__name
     “Department of Software Engineering”@en .

_:954783125769542934 rdf:type mt:place .
_:954783125769542934 mt:place__name
     “Praha”@cs-CZ-singular-nominative .
_:954783125769542934 mt:place__name
     “v Praze”@cs-CZ-singular-locative .
```

3.6 Reification

On of the important features of the RDF is the ability to make statements about statements. This is called reification. It can be used e.g. to specify an author of a statement.

There is no such universal feature in the DataPile. On the other hand the supplementary columns of the PILE table can be viewed as a special case of reification with a fixed set of predicates. The columns contain information about source of the value, its validity period etc.

Unfortunately, expressing reification in RDF is not very compact. It requires using a new blank node and making at least four statements. The identifier of the blank node can be generated from primary key of the PILE table. The primary key contains sequential numeric value.

```
_:568421369754123695 mt:person__name “John Smith” .
_:r65413 rdf:type rdf:Statement .
_:r65413 rdf:subject _:568421369754123695 .
_:r65413 rdf:predicate mt:person__name .
_:r65413 rdf:object “John Smith” .
_:r65413 mt:valid_from “20050703T15:21:49” .
_:r65413 mt:valid_to “20050821T09:35:12” .
```

The example shows a triple stating a name of a person together with triples that give validity period of the statement.

4 The Experimental Implementation

An experimental implementation of the presented transformation has been created and tested on real data.

4.1 Limitations

The implementation does not include direct support for multilingual attributes nor does it support reification.

4.2 The Data

The data for the experiment has been gathered into the DataPile from different information systems at the Charles University in Prague. Variability of these systems provided us with data that have not only complex schema but also greatly vary in their completeness.

Because the implementation does not support reification the data was limited only to records that are considered to be currently valid. Working with historical versions of data requires access to supplementary columns of the PILE table which requires reification. If all of the data was extracted without the supplementary information it would have created multiple attribute values for one attribute of one entity instance without a way to distinguish the valid values from the historical ones.

4.3 The Test Environment

The current implementation of the DataPile uses Oracle Database 10g for storage. The database was running on a dual XEON P4 2.4 GHz with 2GB RAM and SCSI RAID.

The extractor itself was running on a separate machine with four XEON P4 2.5 GHz CPUs with 16GB RAM and SCSI RAID. It accessed the database directly using Oracle Call Interface with thin abstraction layer on top of it.

The performance of the extraction process depends mostly on the performance of the database. Processing of records returned from the database does not require much memory or CPU time.

4.3 The Extraction

The extraction generated a TURTLE file with 26 813 044 RDF triples. We made two runs of the extraction. In the first run the data was sorted by the entity instance

identifier and attribute. The sorting of the data was done by the database system that contains the DataPile. Although it is not required for the transformation to work it can improve performance of further processing of the data and help with debugging.

In the second run the data was not sorted at all.

The sorted version finished in 1738 seconds while the unsorted took 1073 seconds to complete.

5 Conclusion

Even the very basic version of the extraction provided great amount of interesting data. Implementation of a version handling multilingual attributes is planned in the near future.

We plan to use the extracted data in the development of an experimental RDF database that uses a SPARQL language [5]. It will help us test and tune the performance of such database. The data was gathered from systems that are used in practice and so their schema, size and structure represent real requirements of such systems. The test results should tell us how the database would behave when deployed as a basis for large scale information system or a system integrating large heterogeneous data.

References

1. Bednarek D., Obdrzalek D., Yaghob J., Zavoral F.: Data Integration Using DataPile Structure. In proceedings of the 9th East-European Conference on Advances in Database and Information Systems, Tallinn, Estonia, 2005
2. Carroll J. J., Klyne G.: Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, 10 February 2004 http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
3. Beckett D.: Turtle - Terse RDF Triple Language http://www.dajobe.org/2004/01/turtle/
4. Alvestrand H.: Tags for the Identification of Languages http://www.ietf.org/rfc/rfc3066.txt
5. Prud'hommeaux E., Seaborne A.: SPARQL Query Language for RDF, W3C Working Draft, 23 November 2005 http://www.w3.org/TR/2005/WD-rdf-sparql-query-20051123/

Acknowledgement

This research was supported in part by the National programme of research (Information society project 1ET100300419).