Semantic Support for Tables using RDF Record Table

Mari Wigham[1], Hajo Rijgersberg[1], Martine de Vos[2], and Jan Top[1,2]
[1] Wageningen UR, Food and Biobased Research
Wageningen, The Netherlands
{firstname.secondname@wur.nl}

[2] VU University Amsterdam
Amsterdam, The Netherlands
{firstname.prefix.secondname@vu.nl}

Abstract—**Tabular datasets are common in many domains, for example science and engineering. These are often not very well specified, and are therefore hard to understand and use. Semantic standards are available to express the meaning and context of the data. However, present standards have their limitations in expressing heterogeneous datasets with several types of measurements, missing data, and irregular structures. Such datasets are abundant in everyday life. We propose the RDF (Resource Description Framework) Record Table vocabulary for semantically modelling tabular data, as a supplement to the existing RDF Data Cube standard. RDF Record Table has a nested structure of records that contain self-describing observations, and is able to cope with irregular, missing and unexpected data. This allows it to escape the constraints of RDF Data Cube and to model complex data, such as that occurring in science and engineering. We demonstrate our Excel add-in for transforming data into the Record Table format. We propose a general approach to integrating tabular data in RDF, and confirm this approach by implementing integration support in the add-in and evaluating this in industrial use cases. This semantic support for tables helps researchers and data analysts to get the most out of available quantitative data with a minimum of effort.**

Keywords - semantics; table; spreadsheet; e-science, integration.

I. INTRODUCTION

Tabular data are common in many domains, for instance science and engineering. Tools to handle such data, such as spreadsheets, are extremely popular because of their flexibility and ease of use. However, this flexibility often leads to data being ambiguous or even incomprehensible, and their provenance being unknown [1][2][3]. The possibility to immediately proceed to the analysis and visualization of the data can have a negative effect on the quality of the actual data registration in terms of complete and systematic recording. Our work on introducing electronic lab notebooks in the multidisciplinary domain of food science has revealed many issues in data recording in the lab. Annotation of the data is often scarce and ambiguous due to the focus of researchers on the research itself rather than bookkeeping. In addition, large amounts of data are produced by automated measurement equipment in the lab. These devices tend to produce more systematic metadata, but linking data from different sources is as yet difficult and labor intensive. This makes finding, understanding and reusing the data very difficult [4]. As the amount of available data is exploding, it is essential to be able to efficiently locate and reuse existing datasets.

The traditional way to present tabular data is in tables on paper or on a screen. Rows and columns of cells make up their structure, and these cells are filled with simple data types such as numbers, strings or dates. In such a table, an individual recording shows up as a single value in one of the table cells. The associated header cell along the same column or row explains the meaning of this value, for example ‘m (kg)’ for mass measured in kilograms. In datasets found in practice, this header information is often ambiguous and incomplete. In fact, much of the information about the actual observation is frequently left out. This may even be done on purpose, in order to clean the data for presentation or processing. Tables also become more compact if all records contain the same quantities, the same unit of measure and have the same interpretation. In this way, the ‘bare’ numerical or string value in the table cells is separated from the metadata, and is directly visible for comparison and available for numerical computation. Researchers are trained in reading such tables and can usually interpret the meaning of the structure immediately. However, ambiguities in the structure can still arise, for example empty cells may be intended by the author to convey that the content of the previous cell should be repeated, but may cause confusion in a reader.

While the structure is usually easy to interpret, the frequently ambiguous and incomplete content of the headers gives readers more trouble. Abbreviations, ambiguous indications of quantities and units, language differences, jargon and typos all contribute to spreadsheets being frequently incomprehensible to all but the author. After time has passed, even the author may have trouble.

Interpreting such spreadsheets correctly is therefore hard enough for human beings, but next to impossible for a machine. This cuts off an enormous source of potential support for users. With all the computing power at their disposal, they are reduced to browsing through data files to find the one they need, and cutting and pasting data to combine it.

Fortunately, for the further exploitation of datasets, we are not bound to this traditional representation of a table. We can use richer representations to express more contextual information by using semantic technology. Many semantic methods have been developed over the last decades to express tabular datasets in a richer, more flexible manner. The W3C RDF (Resource Description Framework) standard provides a general, graph-based language for describing datasets [5]. RDF Data Cube is a prominent example of an RDF-based standard for expressing tabular datasets [6].

Representing datasets semantically has major advantages. Firstly, the meaning of the measurements is independent of the precise text in a spreadsheet, so that data can be found and understood regardless of typos, abbreviations, local terminology and even different languages. Secondly, the use of semantic concepts makes tables machine readable, meaning that they can be (semi-) automatically processed, from simple unit conversion up to complex computations. Finally, allowable numerical values and units can be defined, making it possible to check or clean the data. Moreover, semantic tables can be used as templates for future observations and experiments.

Initially, we proposed spreadsheet templates to stimulate systematic annotation of research data, but experience has shown that this restricts the creative and essentially unstructured character of scientific research. Therefore, a standard is necessary that facilitates annotation in a way that is flexible enough to accommodate researchers' needs.

Which requirements should a semantic standard meet to facilitate and stimulate structured annotation of tabular data? Firstly, it should be able to annotate the individual data elements, the content. For example, it should be possible to state that 'the mass of this sample is 2.95 grams', 'the city considered is Amsterdam', or 'this event occurred 5 minutes and 6.3 seconds later'. Good scientific recordings contain extensive information about each observation, for example on which object it has been measured, by which method and by whom. The annotation (metadata) of the individual data elements explains them and describes their provenance and relations. The keystone of semantics is the idea of an ontology, a sort of vocabulary that describes shared concepts and the relationships between them. A standard has to build on existing (domain) ontologies in order to facilitate shared understanding of the individual observations.

Secondly, a semantic standard for tabular data should make explicit the structure – the grouping together of scientific observations that collectively form a 'snapshot' of the world. The observations may be combined because they are generated in one experiment, using the same experimental protocol or by a single apparatus, or for a multitude of other reasons. A collection of snapshots, or *records*, is used to detect patterns, similarities or correlations.

This grouping is essential for correct interpretation of the data. Within one experiment, the structure of the records is often quite similar. However, when comprehensive recording of all possibly relevant effects is required, datasets can be less homogeneous and well-formed. This holds for datasets that combine observations from different origins, in particular. Moreover, exact science typically deals with quantities having diverse scales, units and other specifications; values may be missing or occasionally additional measurements are available. Consider for example research that combines input from a number of labs around the world. Some of them have recorded the environmental temperature in degrees Fahrenheit and others in degrees Celsius. One lab has not measured temperature at all. Semantic standards should allow these variations and at the same time provide enough structure to preserve the meaning of the data.

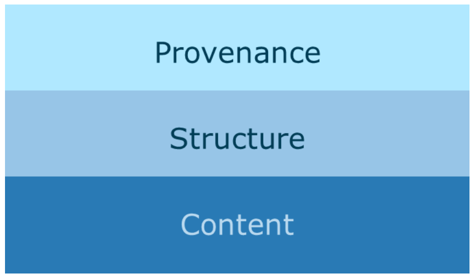


Figure 1: The three components of a semantic standard for tabular data

Thirdly, a semantic standard for tabular data must make it possible to link to provenance information, to indicate where the data came from. Well-publicized cases of fraud in scientific research make the traceability of data a central concern to many research institutes. Fig.1 shows the three components of a semantic standard for tabular data.

Finally, the semantic standard must be flexible enough to accommodate the variations present in scientific data, and be implemented in tools already in use by researchers, in order to harmonize with their research work, rather than distracting from it.

In this paper, we discuss RDF Record Table, a format that is sufficiently rich and flexible to handle complex datasets, such as those often found in science and engineering. RDF Record Table was first introduced in [1]. In this paper, we will expand on the description of the model and discuss its benefits in more detail. In Section II, we first briefly describe existing approaches and tools for modelling tabular data in RDF. In Section III, we go into more detail on the RDF Data Cube vocabulary. This is a recommended W3C standard for multidimensional tables. To be able to handle more heterogeneous datasets, we propose RDF Record Table in Section IV, as a supplement to RDF Data Cube. RDF Record Table uses self-contained observations and recursive records. In Section V, we describe how we can reduce redundancy and include header information in the model by allowing cells to refer to other, similar cells. Section VI discusses the differences between RDF Data Cube and RDF Record Table, in particular with reference to specific challenges faced in scientific data. This is followed by a description of a first implementation for annotating and transforming spreadsheet data to RDF Record Table in Microsoft Excel in Section VII. We then discuss how RDF Record Table makes it easier to integrate data in Section VIII. We describe our approach to data integration, and explain how this approach is implemented in the Rosanne add-in. The approach is validated in a number of use cases in Section IX. Finally, we conclude in Section X, also listing a number of open issues.

II. RELATED WORK

Many methods take the relational database approach when they convert tables or databases into an RDF-based representation [7]-[9]. They assume that a table consists of a header row defining variables, and other rows that contain strings or numbers representing the value of the variable in the same column. In general, they do not support more complex structures. All columns are translated into RDF properties of a single object. At this point, no other metadata is available than which is given in the header and data cells, the information implied by the table structure is lost. For simple data this may suffice, however, problems quickly arise with more complex data. For example, repeated measurements of the same property will simply produce triples with two different values, without the context that would allow an understanding of why the measurements are different (different time, different apparatus, etc.). The information contained in one column may also be necessary to correctly interpret the information in other columns. For example, based on the price information of two types of cheese, you might conclude that the cheese with the lower price is cheaper. However, the amount of cheese, recorded in a different column, could be completely different. Tables also frequently contain information that is not a property of the same single object, for example, the temperature of the room in which the density of the sample was measured. Finally, removing all indication of table and cells by converting only the data to triples, removes the ability to convey provenance information about the data.

A richer format is defined by the RDF Data Cube vocabulary [6], a recommended W3C standard. This vocabulary has been developed in the context of statistical data in social sciences and policy studies, but is also being applied in other areas [10]. Information about the meaning of the data is expressed by linking to concepts from other ontologies, most typically the SDMX vocabulary [11]. These individual data observations are stored in a multi-dimensional hypercube structure to preserve the relationship between the measured values and the dimensions along which they vary, such as time, location, gender, etc. Metadata can be linked to individual observations, parts of datasets, or whole datasets.

There are various tools that have been developed for converting tabular data into RDF in general, or RDF Data Cube in particular. The EU CODE project [12] developed the CODE platform, which extracts tabular data from PDFs, csv-based documents or existing RDF repositories and converts it to RDF Data Cubes. These cubes can then be visualized. The Tabels (sic) project [13] attempts to discover the data structures in tabular data and transform these to RDF Data Cube. TabLinker [14] and RightField [15] assist the user in annotating their numerical data, which is then converted to RDF Data Cube, in the case of TabLinker. CSV2Data Cube [16] helps the user to configure dimensions and attributes from their CSV file. It then transforms the data to RDF Data Cube. The OpenCube toolkit [17] [17]from the EU OpenCube project [18] allows relational data and csv/tsv files to be converted to RDF Data Cube. These cubes can then be visualized and also submitted to statistical analysis. Tools for visualization, slicing and validation of RDF Data Cube fall outside the scope of this work.

The available tools are mostly directed at the domain of statistical data and, with the exception of RightField (which does not handle table structure), appear to be limited to simple table structures. Statistical data is, as a rule, much more uniform and regular than scientific or engineering data, which can have quite complex table structures.

All of these tools are separate to the tools that are usually used by researchers in the course of their work (with the exception of RightField, which generates templates that are used in Excel). This requires researchers to interrupt their workflow in order to carry out data documentation. This can be a barrier for researchers.

We have found one incidence of related work on representing more complex, irregular data in RDF [19] investigated linked Data Cubes for clinical data. Some of the difficulties they experienced could be solved by augmenting RDF Data Cube with constructs from other vocabularies, others remained unsolved.

Whereas RDF Data Cube and other standards define the structure and context of tabular data, they are not intended for expressing provenance of data on the web. However, they do provide identifiers for the data, which can be linked to a description of the provenance of that data. For that purpose, additional vocabularies have been developed. The W3C-standard PROV is becoming increasingly popular for this purpose [21]. It describes the origins of any type of data, helping the user to evaluate how appropriate and trustworthy the data is for a particular use. PROV basically says that a `prov:Agent` performs a `prov:Activity`, in which he uses or generates a `prov:Entity`. Tables, records, slices and individual measurements can all be seen as subclasses of `prov:Entity`. The previously defined Dublin Core Terms [13] vocabulary complements the PROV model with detailed concepts about publications and authorship.

We wish to develop a standard for tabular data that can handle the sort of complex, irregular data that is found in many practical situations. This standard will be able to be linked to the PROV standard and will be implemented in tools that researchers already use in their daily work.

III. RDF DATA CUBE

RDF Data Cube organizes observations as multidimensional datasets. Each observation is a point in n-dimensional space, defined by the associated values of the *dimensions*. Typical dimensions in RDF Data Cube are ‘time’, ‘area’ and ‘gender’. Each observation contains one or more *measures*, for example ‘life expectancy = 83.5’.

Table I: Life expectancy data in different regions over time

	2004-2006		2005-2007		2006-2008	
	Male	***Female***	***Male***	***Female***	***Male***	***Female***
Newport	76.7	80.7	77.1	80.9	77.0	81.5
Cardiff	78.7	83.3	78.6	83.7	78.7	83.4
Monmouthshire	76.6	81.3	76.5	81.5	76.6	81.7
Merthyr Tydfil	75.5	79.1	75.5	79.4	74.9	79.6

Observations, measures and dimensions can have *attributes* that provide additional information about them, for example the unit of measure used. A separate section of an RDF Data Cube defines its *structure*; this section can be used as a template for future observations. Another section gives information for external reference to the entire dataset.

In its normalized form, each observation in a data cube contains all its dimensional values. One way to reduce redundancy is by moving shared attributes to the structure definition section. Further reduction can be obtained by introducing 'slices'. A slice is a lower-dimensional representation, which also serves as a proposed interpretation of the dataset. Moreover, one can refer to a slice as an independent entity. This allows easy access to predefined views of the data.

In order to group together observations that do not fulfil the requirements of a slice, the concept of ObservationGroup is defined in RDF Data Cube. This allows any observations, even from different datasets, to be grouped together.

Table I shows the example table that the RDF Data Cube definition uses to explain the vocabulary [6]. The full RDF Data Cube model of Table I is available for viewing at [6].

The RDF Data Cube vocabulary is very well suited for modelling well-formed, complete datasets such as are produced by statistics offices. Software tools are available to provide useful views of the data. However, these advantages are the result of some restrictions on the data. RDF Data Cube is intended for describing 'well-formed' datasets. As a result, several constraints are placed on the data, for example that each observation must have a value for every measure. For example, if for one measurement in the example it is not known whether this person is a man or a woman, then this data point cannot be included in the model. Another assumption is that the multidimensional structure is a regular (hyper)cube, not permitting rows with varying length for a single dimension. If we know the standard deviation of the life expectancy value for Cardiff and Newport, but not the other regions, we cannot add this to the above Table I.

RDF Data Cube has two alternative ways to handle datasets with more than one measure, which cannot be used simultaneously. In the *multiple measures* approach one observation can contain more than one measured quantity. However, all quantities must have the same attributes, for example, the same type and unit of measure. This rules out this approach for many exact science applications. The second approach restricts observations to having a single measured value. It allows a dataset to carry multiple measures by adding an extra dimension, a measure dimension. This turns a measured value into a kind of semi-dimension.

Another characteristic of RDF Data Cube is that it makes extensive use of properties (rather than classes) as its main organizing mechanism. The design introduces many different types of properties. It is questionable whether these different properties are needed to express the meaning of the data. They make the design of a model rather complex.

As datasets, slices, ObservationGroups and observations all have unique identifiers in RDF Data Cube, they can all be referred to by a provenance model, enabling the provenance of the data to be traced.

RDF Data Cube is the only semantic standard currently available which explicitly and thoroughly models the structure of tabular data.

IV. RDF RECORD TABLE

Experience with researchers over the past ten years has confronted us with many different datasets. Many of them are contained in spreadsheets and data analysis tools such as Matlab [22], SPSS [23] and R[24]. Inspired by other initiatives to annotate datasets using RDF, we have devised an approach that can work in the tools commonly used by researchers and at the same time support rich annotation. This approach has at its heart a model for tabular data called RDF Record Table.

The RDF Record Table vocabulary is intended for recording original and processed data across all domains, including science and engineering in particular. It is based on the observation that the common two-dimensional table in reports and spreadsheets is a restricted representation of a more general graph-based table model. A human reader of a table in a report or spreadsheet implicitly combines his or her interpretation of the text in individual table cells with the visual inspection of the table layout (topography, coloring, typesetting, etc.). This forces authors of tables to express two types of information in a two-dimensional format that it is not ideally suited for, viz., (i) nesting of records and (ii) describing metadata. In this section we show how the RDF Record Table model deals with these issues by supporting recursive nesting of records and by enriching data elements with metadata. In the next section, we will show how the model supports sharing of metadata between multiple data elements. RDF Record Table models the structure of tables in terms of cells and records (see Fig. 1, using rec: as a prefix for the RDF Record Table namespace). A *cell* contains a statement about an entity or the property of an entity, such as 'the temperature of this object measured by a pt-sensor is 36.5C' or 'this milk sample is from batch 20140612YTU'. A *record* combines cells in a group, thus conveying the assumption that in some way the observations are related - in time, location, subject, conditions, or in another way. This assumption can be made when setting up a new experiment, but also when existing data are combined. It is similar to the ObservationGroup concept in RDF DataCube, but in RDF Record Table it is a core element rather than an optional extra. Scientific and

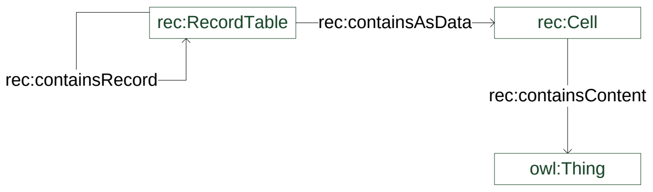


Figure 2: Basic RDF Record Table schema

engineering data are grouped and regrouped continuously to investigate hypothesized correlations and causalities. We submit therefore that the structure of the data should be flexible and based around the groupings chosen by the researcher. To express composite structures, in RDF Record Table any record can recursively contain sub-records, which again are of the type `rec:RecordTable`. This means that we do not make a distinction between the concept table and the concept record. After all, both are simply groupings of data. For example, an experiment may observe multiple samples at one fixed temperature. For each sample its viscosity, composition and mass are measured over time. This means that the entire dataset consists of a Record Table that at its highest level contains (i) the observed temperature and (ii) a sub-record for each sample. Each sub-record in turn contains the sample identifier and sub-records that describe viscosity, composition and mass for that sample measured at a point in time. In the most explicit form, all sub-records are expanded into non-nested records. In this example, the top level Record Table only contains sub-records, each of them stating the observed temperature, time point, sample id and the other measured properties.

RDF Record Table is shown in Fig. 2. In Turtle format, it is defined as follows.

```
rec:RecordTable  a   rdfs:Class ;
        rdfs:subClassOf  prov:Entity .

rec:Cell   a   rdfs:Class ;
        rdfs:subClassOf  prov:Entity .

rec:containsAsData  a   owl:ObjectProperty ;
        rdfs:domain  rec:RecordTable ;
        rdfs:range   rec:Cell .

rec:containsContent  a  owl:ObjectProperty ;
        rdfs:domain  rec:Cell ;
        rdfs:range   owl:Thing .
```

The next question is how the cells in the nested records can contain the actual observed values in such a way that they can be properly understood both by human users and machines. From the inspection of many tables used in practice, we see that two types of observations frequently occur: (i) *identified entities* and (ii) *properties measured on a scale*. Examples of *identified entities* are ‘sample XY876b’, ‘Newport’ and ‘Peter’. Quantities such as ‘length’, ‘mass’, and ‘temperature’ are examples of

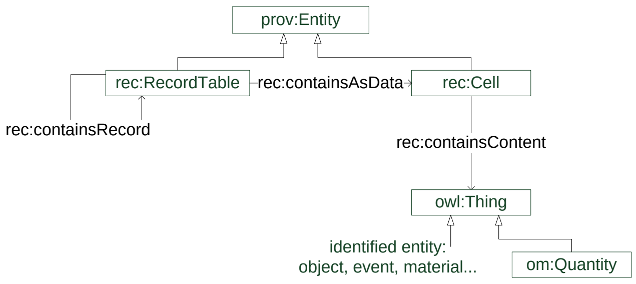


Figure 3: RDF Record Table expressing domain and provenance information

properties measured on a scale. These two types are not formally part of the RDF Record Table model, which allows any ‘Thing’ to be in a cell. However, we propose this distinction as a best practice that works in many cases. Fig. 3 shows how quantities and identified entities fit into the RDF Record Table model.

In traditional tables, identified entities are typically represented by a human readable identifier, and an explanation of the entity type in the associated header cell. For example, ‘Peter’ is a unique name for an entity of type ‘Author’. RDF Record Table uses externally available domain ontologies to express all that is needed to know about such an entity by pointing to the respective instance in an RDFS/OWL schema. For modelling Table I we have chosen to view instances of ‘Area’ and ‘Period’, such as 2004-2006, as identified entities since they are not supposed to be read as nominal or even numerical values.

For the other type of observation, a *property measured on a scale*, RDF Record Table uses ontologies that define quantitative or qualitative values defined on a scale, possibly with units of measure. In Table I, ‘Sex’ and ‘Life Expectancy’ are typical measured properties, one on a nominal scale and the other on a rational scale with unit ‘Year’. In our work we use OM (Ontology of units of Measure and related concepts) [25] for expressing quantitative measurements. OM contains a large number of quantities and units of measure suited to scientific and engineering datasets. It also provides the necessary properties for linking the quantities, domain concepts and units. However, other ontologies such as QUDT [26] and SDMX [11] can be used equally well. The measured quantities can be properties of the observed entities in the table, but do not need to be related to anything specific. For example, in Table I, the life expectancy measured is that of people in the associated geographical region. On the other hand, ‘the time of day’ is usually not connected to a specific entity (except for example to a ‘time zone’ that relates to a geographical area).

This division into *identified entities* and *properties measured on a scale* is highly useful, as it relates to the type of data handling that is typically applied to data from each category. Measured properties are usually subject to numerical processing, and require units of measure. Identified entities on the other hand may be used as identifiers on which, for example, different tables can be joined. While this distinction can assist processing, it does not limit it – for example, tables may also be joined on numerical values, if the user wishes.

Finally, by making `rec:RecordTable` and `rec:Cell` subclasses of `prov:Entity` we ensure that all provenance information can be expressed for individual measurements, records and tables. For example, the relation `prov:wasDerivedFrom` between two cells tells us that the quantity in one cell depends on the value of the quantity in the other cell.

To illustrate the use of the RDF Record Table format, we show how the cells with values 76.7 and 83.3 in Table I are contained in the table. We see that the first level of nesting defines four records (:o1, :o2, :o3, :o4), one for each region. We use the ontology for geographic areas (as *identified entities*) that was also used in the RDF Data Cube example [6]. The next level specifies the three time periods, again using instances that were also used in the data cube example. At the third level of sub-records, we register two properties measured on a scale, viz., ‘sex’ and ‘life expectancy’. For indicating the variable ‘sex’, we use an sdmx-code, as in the data cube; to illustrate the use of OM [25], we use the concept `om:Duration` from that ontology to describe ‘life expectancy’. The value of a quantity in OM is of the type `om:Measure`, which is a combination of a numerical value and a unit (or scale).

```
:dataset1  a  rec:RecordTable ;
   rec:containsRecord :o1 , :o2 , :o3 , :o4 .

:o1  a   rec:RecordTable ;
   rec:containsAsData :cell_newport ;
   rec:containsRecord :o11 , :o12 , :o13 .


:cell_newport  a     rec:Cell ;
    rec:containsContent  ex-geo:newport_00pr .


:o11  a   rec:RecordTable ;
   rec:containsAsData :cell_period_2004_2006 ;
   rec:containsRecord :o111 , :o112 .

:o111  a  rec:RecordTable ;
   rec:containsAsData  :cell_sex-M ,
        :cell_lifeExpectancy_76_7YR .

:cell_sex-M  a rec:Cell ;
   rec:containsContent  sdmx-code:sex-M .

:cell_lifeExpectancy_76_7YR  a rec:Cell ;
   rec:containsContent :lifeExpectancy_76_7YR ;

:lifeExpectancy_76_7YR a om:Duration ;
   om:value :_76_7YR .

:_76_7YR  a  om:Measure ;
   om:numerical_value “76.7”^^xsd:string ;
   om:unit_of_measure_or_measurement_scale om:year
.

...
```

```
:o2  a   rec:RecordTable ;
   rec:containsAsData :cell_cardiff  ;
   rec:containsRecord :o21 , :o22 , :o23 .

:cell_cardiff  a   rec:Cell ;
   rec:containsContent  ex-geo:cardiff_00pt.


:o21  a   rec:RecordTable ;
   rec:containsAsData :cell_period_2004_2006 ;
   rec:containsRecord :o211 , :o212 .

...

:o212  a  rec:RecordTable ;
   rec:containsAsData :cell_sex-F ,
        :lifeExpectancy_83_3YR  .

:cell_sex-F  a   :Cell ;
    rec:containsContent  sdmx-code:sex-F .

:cell_lifeExpectancy_83_3YR   a  rec:Cell ;
   rec:containsContent :lifeExpectancy_83_3YR .


:lifeExpectancy_83_3YR a om:Duration ;
   om:value :_83_3YR .

:_83_3YR  a  om:Measure ;
   om:numerical_value “83.3”^^xsd:string ;
   om:unit_of_measure_or_measurement_scale  om:year
.
```

To show the flexibility of the RDF Record Table model, we now show how a completely different type of measurement can be added to the above definitions, without changing anything in the previously modelled records and cells. In Table I, we add ‘the measured average weight of the inhabitants of this region’ to an existing record (`:o341`) using the OM quantity `om:mass`. In addition, we can switch to a value for ‘life expectancy’ measured in months rather than years for one single observation (74.9 years). The result is as follows:

```
:o431   a   :RecordTable ;
    rec:containsAsData  :cell_sex-M ,
         :cell_lifeExpectancy_898MONTH ,
         :cell_mass_71kg .

:cell_lifeExpectancy_898MONTH  a  rec:Cell ;
    rec:containsContent :lifeExpectancy_898MONTH .

:lifeExpectancy_898MONTH  a   m:Duration ;
    om:value  :_898MONTH .

:_898MONTH  a    om:Measure ;
    om:numerical_value  "898"^^xsd:string ;
    om:unit_of_measure_or_measurement_scale
             om:Month .

:cell_mass_71kg  a    rec:Cell ;
    rec:containsContent  :mass_71_kg .

:mass_71_kg  a  om:Mass ; om:value  :_71_kg .

:_71_kg  a  om:Measure ;
    om:numerical_value  "71"^^xsd:string ;
    om:unit_of_measure_or_measurement_scale
              om:kilogram .
```

Note that so far we assume that each cell is entirely self-describing; it contains all the necessary information to know what kind of data it represents. Where data cells are similar, it is possible to use one description for many cells. We will discuss this in the next section.

V. Handling similar data elements in RDF Record Table

As stated before, traditional two-dimensional representations of tables express descriptive information about the data in the table headers. In their most basic form they form a single row at the top of a table, but much more complicated header structures occur commonly. Each header cell covers a range of data cells, typically shown in the column under the respective header.

In practice the distinction between header items and data items is not always clear. For example, in Table II, it is possible to view the top three rows and the left column as headers. In fact, only "Life Expectancy", "Period", "Area" and "Sex" are true headers, as they only supply descriptive information about the data. The other 'header' cells, such as "Male", and "2004-2006", actually supply different data values for one data type. This style of table, where the 'header' contains data values, is often called a 'pivoted table', as it can be produced by pivoting a 'flat table', where the header only contains descriptions of the data. Pivoted tables can give extra insight into data by grouping together data for which one field always has the same value, for example all data relating to "Cardiff". Any RDF Data Cube with more than one dimension is in the style of a pivoted table, the effect of choosing between a dimension and a measure is to select the measurements on which the data will be pivoted. A pivoted table can be 'unpivoted' by adding the data elements from the header to each record.

The RDF Record Table model defined in the previous section assumes that all data in the table cells are entirely self-descriptive. Each data element describes what kind of data it represents. For example, 't_{final} = 42 sec' expressed using OM concepts says that an activity has ended after 42 seconds. Traditional tables in reports and spreadsheets usually summarize this explaining information 't_{final} (sec)' in a table header, separately from the numerical '42', to make the table readable for humans and fit for numerical analysis. In RDF Record Table, in principle, we can do without such headers, as all this information is available in the data cells; in the above example the data cell would be linked to the concepts 'time' and 'seconds'. In the case of a pivoted table, the 'header' information is simply another data value

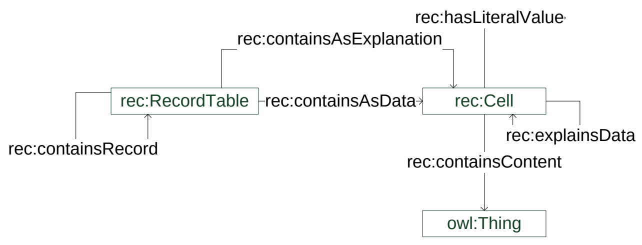


Figure 4: RDF Record Table with 'header' cells.

in the Record Table, with a nested Record Table containing the information that falls under the 'header'.

In practice, many data items in a single experiment are similar in some way – they refer to the same type of parameter, or play the same role in an experiment. We submit that the way that traditional tables express this, is inherently limited due their two-dimensional character. In RDF Record Table we look at table header cells in a generalized manner, independent of their usual two-dimensional representations. We assume that a 'header' cell explains a set of similar data items. It provides metadata that is not expressed by the individual data items.

There can be four reasons to put this information in header cells rather than in self-contained data cells. First, header cells can specify the type of measurement without giving actually observed values; they act as a prescriptive template for an experiment or for data analysis. Second, using header items for metadata is a way to remove redundancy and to achieve a significant reduction in the physical size of a dataset. For example, suppose that the temperature of an object has been measured over time. We can state in a header item that we have measured temperature, measured in kelvin, on a given object. In the corresponding data items we only have to state the numerical values. This takes much less space, but still allows us to regenerate the self-contained values for all data elements (if we know how to link the numbers to the reconstructed instances). Third, cells of different types may play the same role in a table. For example, a column containing numerical measurements may also include the entry 'measurement failed'. This cell clearly has a different type, but should still be grouped under the column – it plays the same role as the other measurements. Finally, the header cell itself may contain additional, informal information. For example, the header cell 'Area' may have contained the string 'Area, as per 2001 boundaries", or "Life Expectancy" may have been written using the Dutch word "Levensverwachting". While ideally this sort of information would also be modelled semantically, in practice this is not the case. If the header cell is modelled separately to the data cells, then its text content can be preserved exactly as it was, keeping any informal information and also making the table representation more familiar to the user.

Table II: Example extended with headers

Life Expectancy (years)							
Period		***2004-2006***		***2005-2007***		***2006-2008***	
Area	***Sex***	***Male***	***Female***	***Male***	***-***	***Male***	***Female***
Newport		76.7	80.7	77.1	80.9	77.0	81.5
Cardiff		78.7	83.3	-	83.7	78.7	83.4
Monmouthshire		76.6	81.3	76.5	-	76.6	81.7
Merthyr Tydfil		75.5	79.1	75.5	79.4	74.9	79.6

In RDF Record Table we do not define different types of cells for data and metadata since their internal structure is the same. Instead, we use the property `rec:explainsData` to have some cells act as metadata cells (see Fig. 4). Such an explaining cell contains an instance (or class) of a phenomenon or a quantity, acting as a template for other cells. The associated data cell in that case only needs to provide a numerical or string value (through `rec:hasLiteralValue`), all other information is derived from the explaining cell.

For example, in Table II, one header field states that we have observed 'Life Expectancy' in 'years'. We construct a cell in the RDF Record Table translation that contains an instance of `om:Duration` with unit 'year' but no numerical value assigned to it. The cell also preserves the string in the original table header by storing 'Life Expectancy (years)' as a literal value. The associated cells only have to provide a numerical value for each measurement and a reference to this header item. Or in another case, the header item 'Area' states that we have observed entities of type `admingeo:UnitaryAuthority`, and the data items express specific areas such as 'Monmouthshire', only using a literal string. When unpacking this compacted version to a fully self-describing model, the software has to match the string 'Monmouthshire' with instances of `admingeo:UnitaryAuthority` to find the proper instance `ex-geo:Monmouthshire`.

Suppose that no interpretation whatsoever is possible given a specific traditional table, except for its structure. In that case, we can translate this table directly into an RDF Record Table with only literal values, using only the property `rec:hasLiteralValue` for both header items and data items. If we can also map the rows and columns directly to records, data cells and explaining cells, this would be the least semantically rich representation possible. Once translated to RDF Record Table, we can process this information and possibly add more semantics to it. RDF Record Table allows each data cell to have its own specification, overruling the information in the header item.

In the previous section we have shown an example in which a single cell measured 'life expectancy' in 'months', whereas all others were measured in 'years'. In that case, we could have used an RDF Record Table model with an explaining cell that states that in principle all values are in years. The single cell that uses 'months' as a unit overrules this general statement for that specific cell.

Regenerating an RDF Record Table with semantically self-contained data items is possible if we know how to relate the information in partially specified data cells to the associated explaining items. When using for example the OM quantity `om:Duration` with unit `om:year` in a header cell, we know that a numerical value in an associated data cell specifies the 'numerical value' property of the Measure of this quantity. This is the type of interpretation that readers of tables on paper make all the time, but is not obvious for automatic processing. This knowledge has to be incorporated in the software that unpacks a model.

Fig. 4 shows the extended schema for RDF Record Table, including header information using the property `rec:explainsData`. It shows that the class `rec:Cell` can play the role of either header cell or data cell. As a matter of fact, it is possible to use a data cell as a header item. This means that other data items use this particular item to provide their type and other information, while the data items simply provide a numerical or string value.

In Fig. 4, we can see that records can directly indicate which cells play a role as header cells using the property `rec:containsAsExplanation`. For Table II such a listing of explanatory cells would be modelled as follows:

```
:dataset1  a  rec:RecordTable ;
   rec:containsRecord :o1 , :o2 , :o3 , :o4 ;
   rec:containsAsExplanation  :cell_sex ,
       :cell_lifeExpectancy_YR ,
       :cell_period_from_yr_to_yr
       :cell_geographicalArea .
```

The cell explaining the Life Expectancy measurements then refers to the data cells containing those measurements:

```
:cell_lifeExpectancy_YR  a  rec:Cell ;
   rec:explainsData
       :cell_lifeExpectancy_83_3YR ,
       :cell_lifeExpectancy_76_7YR ,
       :cell_lifeExpectancy_898MONTH ;
       :hasLiteralValue  "Life expectancy
(years)"^^xsd:string .
```

Although for reasons of clarity it is attractive to write the explaining cells as a 'header' at the first level of nesting, there is no formal need to do so. Such cells can be placed anywhere in an RDF Record Table model. This is useful when merging data from different sources, where the top level table is not known upfront.

Finally, we note that the property `rec:explainsAsData` is redundant if the respective cells are already used to explain data cells through the property `rec:explainsData`. However, this property can be used to express the structure of an otherwise empty table, which then can serve as a template for new observations or analyses.

VI. Differences between RDF Data Cube and RDF Record Table

The most salient difference between RDF Data Cube and OQR Record Table is the fact that RDF Data Cube sees complex datasets as n-dimensional hypercubes, whereas RDF Record Tables are defined recursively via nesting. This use of n-dimensional hypercubes in RDF Data Cube has a profound impact on the type of data it can be used to model. RDF Data Cube expects a certain kind of dataset:

“At the heart of a statistical dataset is a set of observed values **organized along a group of dimensions**, together with associated metadata” [6]

RDF Data Cube also requires that cubes be well-formed, which requires, among other things, that there be no missing data, and that all measures and dimensions are required for all observations. In short, Data Cube expects a uniformly formed and filled cube, with no extra granularity in some areas that is missing in others, and no extra observations. This is often not the case, particularly when data from different sources is integrated together. In these situations, the integrity constraints can cause problems:

“Some specialised data cubes do not satisfy the integrity constraints, specifying that every qb:DataStructureDefinition must include at least one declared measure (IC-3), that only attributes may be optional (IC-6) and that each individual qb:Observation must have a value for every declared measure (IC-14). These constraints are too restrictive for our Nutrition data cube where the presence or ab-sence of a value for a particular category of food varies according to the subject’s diet. This is a concern for survey questionnaires using previously entered values to determine if a field on a form should be mandatory filled.” [19]

The authors of the above-mentioned paper specifically note the difference in their data from that commonly used in RDF Data Cube.

“The LCDC (Linked Clinical Data Cubes) use of the RDF Data Cube vocabulary is different from the more common use cases [10] primarily because of the unreliable, disparate and longitudinal nature of clinical data” [19]

RDF Record Table, on the other hand, has been specifically developed for researchers and their quantitative data, with extensive input from real-life research data. This data, like the data used by [19], is often far more irregular than most statistical data. In RDF Record Table any record can contain an arbitrary set of measurements, with different types and sub-records. Missing values or varying units of measure or other attributes within a single dataset are no problem. We do not demand completeness or regularity of the data, in the sense that a record can contain any set of entities and properties. This better reflects the reality of datasets in science and engineering, in particular, when datasets from different sources are combined. It can be argued that such datasets can be modelled in RDF Data Cube simply by violating the integrity constraints. However, this is a bad approach to using a standard, and can lead to interoperability problems between tools developed for the standard.

The second major distinction between the two approaches is that RDF Data Cube distinguishes between dimensions and measures, whereas OQR Record Table does not make a priori assumptions about the roles of individual observations. We consider making such decisions to be the task of the data analyst.

We will now further discuss the differences between RDF Data Cube and RDF Record Table in the context of specific challenges that are faced in annotating and integrating real-life data.

1) *Missing data*

According to the integrity constraints for RDF Data Cube, all data must be present. Naturally, even in the well-planned world of statistics bureaus, data is sometimes missing. There is no solution in the RDF Data Cube standard for this. However, the ‘attribute’ concept, which allows metadata to be attached to an observation, is a natural way to indicate missing data. In [10], a simple Boolean attribute is used to indicate when data is missing. It is of course then necessary that tools using the data are aware of this solution and can process it correctly.

When two tables are integrated together, there can be missing data even though both original tables were complete. For example, if one file contains the mass measurements for all dairy products, and another file contains viscosity measurements for all liquid dairy products, then when the two files are integrated together, there will be missing data for the solid dairy products. In order to cope with this in Data Cube, observations would have to be generated for these products, and then marked as ‘missing data’.

In Record Table, there is no constraint requiring data to be present. Therefore, in the event of missing data, the cell can simply be omitted from the record. This fits better with the ethos of RDF.

Table III: Example of an irregularly nested table

			Installed Capacity (in 10 MW)		Operating Water Level (meter)	
Hydropower Stations	Budget (in hundred millions of CNY)	Completion Year	*Realisation level*		*Realisation level*	
			2003 Plan	*Upon Completion*	*2003 Plan*	*Upon Completion*
Twenty-Second	37.76		60	60	519	519
Twenty-First			15	15.5	533	533
Twentieth	79.37	2009	150	175	602	602

2) *Unexpected data*

In a table, a column can often contain an unexpected value. For example, a column of numerical measurements would be expected to contain values such as “1.34, 22452E-10”. However, it is perfectly reasonable that a researcher may note down unexpected values: “<20, negligible, ~5”. These are results that a human reader will be perfectly capable of processing when they appear in a table, but that can confuse a software tool.

In RDF Data Cube the ranges of the dimensions and measures are set. A column that expects to contain decimals, cannot therefore contain exceptions such as we name here. An option would be to store the value in an attribute, however, we regard the storing of data in a metadata field as highly inadvisable.

In RDF Record Table, the role of a cell is separate to its type. A cell containing content with type string, for example, ‘negligible’, can therefore be linked via the relationship `rec:explainsData` to a header containing content with type Mass. This allows the information that the mass is negligible to be stored with the correct data type, so it can be excluded from numerical processing such as aggregation. At the same time the role of the information is clear, which allows the value to be displayed along with the other mass measurements.

3) *“Irregular” nesting*

RDF Data Cube demands that every observation has a value for each dimension:

“Every `qb:Observation` has a value for each dimension declared in its associated `qb:DataStructure-Definition.”[6]`

This means that RDF Data Cube cannot model any tables that do not fulfil this requirement. Regularly nested tables, in which for each observation there is a value for each dimension, can be modelled without problems. However, tables are often partially or irregularly nested. The observations in these tables then do not have a value for each dimension. These data values are not missing as such, the dimension is simply not applicable for part of the data. Table III is a real-life example of such a table.

It is perfectly logical that this information about the constructed dams is stored in one table. However, this table cannot be modelled as one Data Cube, as the Completion Year and Budget observations do not have values for each value of the Realisation Level dimension, because their value is not affected by the Realisation Level. It would have to be split into two Data Cubes, one with Dam name as Dimension, and Completion Year and Budget as Measures (Table IV); and one with Dam name and Realisation Level as Dimensions, and Installed Capacity and Operating Water Level as Measures (Table V). Alternatively, the Budget and Completion Year data could be repeated for each Realisation Level, but this creates the misleading impression that there is a relationship between these data and the Realisation Level.

Either approach requires either a fairly advanced level of understanding from the user, or quite intelligent processing from the data input tool. Breaking up the table into two Data Cubes also loses the implicit relationship between the data, which must then be indicated in metadata or by grouping the Data Cubes in an ObservationGroup. An alternative solution, namely using the void:subset relationship to indicate a link between Data Cubes, was used by [19]. This underlines the need for this sort of nesting in real-life data.

In RDF Record Table, the concepts of Dimension and Measure do not exist. The table can simply be annotated as it stands, and the nesting of Record Tables allows the extra information on Realisation Level to be added in to only the relevant portions of the table. The data is kept together, and the original structure (with all its implicit information) is retained, without need for additional constructs such as ObservationGroup.

4) *Multiple measures*

In the above example, one table had two Measures – Installed Capacity and Operating Water Level. As explained in the section on RDF Data Cube, in such a situation the user must choose between modelling these with multiple measures, or with a measure dimension. For many situations the choice made may not matter in practice; however, the choice must always be made. For novice users

Table IV: The first of the two tables into which Table III must be split when modelled in Data Cube

Hydropower Stations	**Budget (in hundred millions of CNY)**	**Completion Year**
Twenty-Second	37.76	
Twenty-First		
Twentieth	79.37	2009

this can be confusing. When integrating two tables, one of which uses multiple measures, and the other a measure dimension, a conversion will also have to take place before the integration, as the two types may not be mixed in the same dataset, according to the RDF Data Cube specification.

RDF Record Table requires no choice for how to handle multiple measures, as no distinction is made between measures and dimensions. Each record simply has a number of cells, each cell with a single value. Where measured values belong together, such as in the case of a multi-spectral measurement, they can be grouped together in their own Record Table, which can be nested within the larger Record Table.

5) Ease of annotation

Setting up an RDF Data Cube requires a certain level of technical knowledge of the model. While a data entry tool can of course hide away all the complexities of the RDF itself, the user must, at the very least, specify their Dimensions and Measures. For nice, regular examples, such as those given on the RDF Data Cube website, learning how to do this is perhaps not so hard. But for more complex examples, it is asking quite a lot of the user to be able to do this correctly. It is of course possible to choose the approach of having an expert design a template for users to fill in (as in RightField [15]). The users themselves are then not required to understand the model. However, this limits the spontaneity and creativity of the users, they cannot make a simple change such as adding a new data column without needing to apply for a template change.

For RDF Record Table, on the other hand, the user does not need to make the distinction between Measures and Dimensions. All they need to do is to annotate headers with quantities, phenomena or units of measure. As the difference between a quantity and a phenomenon is not dependent on their role in the table, it is quite easy to learn. In the Rosanne tool, which implements RDF Record Table, and which we will discuss in Section VII, even this knowledge is not necessary, as the user simply looks up the annotation they want to apply, based on the name of their item.

6) Ease of integration

In RDF Data Cube, 'tables' and 'records' don't exist, the data is all merged into the hypercube. To integrate data from two different data cubes together on a given JOIN field, first the dimension to be used as the JOIN field must be chosen, optionally values of additional dimensions must be specified to select a section of the data, and finally the desired measures must be selected. For example, for the life expectancy table, we could specify Region as the JOIN field, the dimension value 2004-2006 to select that time frame, and then for the measure the only option is Life Expectancy. Given a table of average weight for the same time frame and region, we could then select the measure Average Weight, and so produce a table showing the average weight and life expectancy for all regions in the time period 2004-2006. Inherent to the integrity constraints of Data Cube is that we could not have done this if the average weight was only available for half the regions, without an extra step to generate empty 'missing data' observations for the other regions. Similarly, if the data we had available on average weight was not split into male and female, the integration could not occur, as the gender dimension would be missing in part of the integrated table. The available options are limited by the constraints placed on the data.

Table V: The second of the two tables into which Table III must be split when modelled in Data Cube

Installed Capacity (in 10 MW)			**Operating Water Level (meter)**	
Hydropower Stations	***Realisation level***		***Realisation level***	
	2003	***Upon Completion***	***2003***	***Upon Completion***
Twenty-First	60	60	519	519
Twenty-Second	15	15.5	533	533
Twentieth	150	175	602	602

RDF Record Table is built around tables and records. We integrate using SPARQL [27], a semantic web querying language. To carry out the same integration as above, we select all records containing Region from the desired tables, and select the Life Expectancy and Average Weight cells. To define the time period, we set the value of that entity to 2004-2006 (remember, we have assumed that these time periods are identified entities). Missing data can be accounted for using the SPARQL OPTIONAL keyword (see Section VIII), and we can still integrate average weight information even if it is not split into male and female (the weight information is organized per region and time period, with additional nested tables containing the life expectancy per gender group). In addition to this, if desired we could join tables based on a numerical value, such as the value of the life expectancy, instead of an identifier, such as region. The distinction between Phenomenon and Quantity can guide in the selection of a join variable, but it is not required that the join variable be a Phenomenon. A table may consist solely of numerical values, if desired, and for scientific analysis such as finding correlations between variables, such a table is perfectly reasonable. RDF Data Cubes, on the other hand, must contain a dimension, and if the measured value is turned into a dimension, then it may only take predefined values. There is much more freedom in how to integrate the data when using RDF Record Table.

7) *Ease of searching and viewing*

RDF Data Cube includes a data structure definition. This immediately supplies information about the expected elements in the data and its structure, making search very easy. The concept of slices allows for a particular view on the data to be quickly obtained, and the concept of dimensions makes the definition and selection of a particular slice very simple. The regularity of the data also aids search, if all integrity constraints are fulfilled then the search does not need to handle missing or optional data.

RDF Record Table does not require a data structure definition. This increases flexibility, but means that the data must first be searched to discover what types of observations are available. The use of `rec:ContainsAsExplanation` to indicate headers at the top level of the table can help make this search quicker. As there is no slice concept, pre-prepared views cannot be provided, and the absence of a Dimension concept means that selection of a given ‘slice’ is more complex, requiring constraints in the query. As the data is not constrained to be regular, the search query must also handle missing data and nested Record Tables. This adds to the complexity and probably reduces the speed of search.

8) *Flexibility of data analysis*

The requirement to choose *a-priori* between dimensions and measures is useful in fields such as standard statistics, where it is very clear what data is to be gathered. Defining dimensions and measures makes it easier to gather the data, and easier to define particular views. This requirement is, however, often problematic, particularly in research, where it is often not clear in advance what is going to be measured. Rather than having a specific measure that is influenced by certain dimensions (time, place, gender), it is often the task of a scientific study to determine what the relationship is between various measurements. Depending on the purpose of the study, the same measurement may assume the role of a cause or a consequence. Rather than assuming some causal order between quantities, therefore, it is more appropriate to simply state that they have been observed together. This is particularly the case for in-vivo studies, where it is much more common to observe various variables and try to discover their relationship, than to vary one particular variable to discover its effect, as it is often impossible to set the values of certain variables to fixed points (as is the requirement for dimensions).

We conclude that RDF Record Table can be viewed as a generalized RDF Data Cube, making fewer assumptions about the regularity and completeness of the data. If a dataset that was originally drafted as an RDF Record Table meets certain requirements, it is in principle possible to automatically transform it into an RDF Data Cube. Any dataset expressed in RDF Data Cube, on the other hand, can be modeled as RDF Record Table. This has the great advantage of allowing data in the Record Table format to still take advantage of all the tools available for Data Cube, where the data meets the Data Cube requirements. It is quite conceivable that both models could be used in the course of the same study. RDF Record Table is appropriate during the research process when data can be incomplete, the researchers are still building their understanding of the data and the role of the different factors, and diverse datasets are being integrated together. RDF Record Table then gives the researchers maximum flexibility to carry out their work without worrying about constraints, dimensions and measures. RDF Data Cube is appropriate when the data has been processed and cleaned up, and the roles of dimensions and measures are clear. RDF Data Cube then allows the researchers to take advantage of the available Data Cube tools for visualization, and to define slices of their data to make consumption and publication easier.

VII. ANNOTATION IMPLEMENTATION

In the following sections, we discuss two tasks that benefit from the definition from a formal model of tables and RDF Record Table in particular. In this section, we discuss how annotation of two-dimensional tables can be done in practice. This annotation is a necessary precursor for the transformation of the two-dimensional table to the RDF Record Table model. In the next section, we discuss practical support for the data integration task.

A good model of tabular data is useless if the data cannot easily be input. Given the popularity of the classic table format in tools such as spreadsheets, it should be possible to use these for data entry and then construct semantic datasets from there. In order to make this process as easy as possible, it should fit into existing work procedures and tools, and minimize additional effort by the user. Since Microsoft Excel is extremely popular, we have implemented the RDF Record Table model as an add-in for Excel, called Rosanne [25]. Rosanne supports engineers and scientists in creating semantic tables (as yet simple, non-nested, non-pivoted tables, i.e., rectangular with one header row or column, with no data values in the header). Similar functionality for the RDF Data Cube has been implemented in TabLinker [14]; however, this is a standalone tool which cannot be accessed from within Excel. Rosanne allows users to enter their data in a simple table format. Rosanne then uses OM (Ontology of units of Measure and related concepts) [14] to assist users in adding relevant quantities and units of measure to the table. In addition, other domain-specific ontologies are available for annotating identified entities in the table, such as samples, objects, locations, etc.

Support for table annotation takes two slightly different forms. In the first case, when creating and filling new, initially empty tables, the user must be assisted in selecting and assigning the right concepts and constructing the right layout. Rosanne supports this task of creating and semantically enriching tables from scratch. It does not confront the user with the Record Table model, nor does the user have to have any knowledge of ontologies. The user sets up a table by simply drawing areas in the spreadsheet. Next, the user selects the concepts they want from dropdown lists showing the user-friendly labels from the ontologies. The URIs (Uniform Resource Identifiers) for the ontology concepts are then stored in the Record Table model.

The second form of annotation is when existing datasets have to be semantically enriched. Rosanne can automatically annotate existing data with units and quantities from OM, based on heuristics [28]. This does not always produce accurate results, but saves time for the user by creating an initial annotation that can be corrected where necessary.

In addition to annotating the content of the cells in tables, a tool that handles spreadsheet tables also has to make an interpretation of the structure of a table. It needs to translate the two-dimensional form into the graph-based RDF Record Table model. Human readers can usually quickly combine layout and text in tables to make the proper interpretation. However, this is not a trivial task to automate since it depends on implicit knowledge. For the current, simple form which we support in Rosanne, an indication of the table and header areas by the user, combined with some basic assumptions in the software, suffice for the majority of tables. However, for more complex structures, more advanced processing is required. We now discuss some heuristics that could potentially be applied to make this translation.

Automatic interpretation of two dimensional tables could be facilitated by making a number of choices and assumptions on the interpretation of the table layout [29]. An important assumption, for example, is that two dimensional tables consist of rectangular blocks with cells that belong to the same semantic category, for example, they are of the same type or they can all be related to a single concept. The measured values of observations, i.e., usually numerical values of type 'float', are often grouped together in one or more blocks. In the example table on hydropower, this is the block in the lower right corner. The blocks adjacent to these float blocks, are usually of type 'string' and provide contextual information on the cells in the float blocks. In the example table, these are the two upper rows, and the column on the left side. These string blocks either represent the quantity that is measured, or the phenomenon of which that quantitative property is measured.

Another assumption is that every observation in a table can be related to a quantity and a phenomenon in the nearest string block. The string cells describing quantities can usually be recognized by the associated units of measure. Automatic recognition of quantities and units of measure can be supported by using an ontology like OM [25], and heuristics such as those used in [28], for example that units are often placed between brackets ‘Mass (kg)’. The string cells describing phenomena are usually located across from the quantity cells.

In the example table, the measure '37.76' can be related to the quantity 'Budget' and to the phenomenon 'Twenty-Second in the Lancang River Cascade'. If an observation can be associated to multiple quantities or phenomena, this could indicate that the corresponding table has a nested structure. In the example table, the measure '60' can be related to the quantity 'Installed Capacity' and to the phenomenon 'Twenty-Second in the Lancang River Cascade', but also to the phenomenon 'Realisation level', indicating nesting.

The string blocks in two dimensional tables are often called table headers, based on their position in the table. However, in RDF Record Table header cells are defined based on their role as descriptive item. Translation from a header cell in a two dimensional table to a header cell in RDF Record Table is therefore not straightforward.

Headers in two dimensional tables often contain a series of instances of phenomena or quantities. These are in fact data values (see section V) and the corresponding cells should therefore be modeled as data items in RDF Record Table. The actual header, i.e., descriptive, item in RDF Record Table is the parent class of these instances. Automatic recognition of these parent classes can be supported by using selected ontologies, for example OM for quantities and a domain vocabulary for phenomena.

The abovementioned assumptions can be used as indication of the composition of records, and properties and roles of observations when translating a two dimensional table into an RDF Record Table. However, science and engineering tables can have complex structures that are difficult to interpret in a fully automated way. A possible solution would be to develop an interactive tool. With such a tool, the majority of the interpretation would still be performed automatically, but user input is required for checking and refining the results.

VIII. Integration of annotated data

Having discussed the annotation task in some detail, we now move to another important task for data handling, namely *integration* of data. Scientific research regularly requires data to be combined from different sources. This may be as simple as merging two different tables from the same experiment, or as complex as integrating multiple tables, each from a different research group at a different time. Integrating these data allows researchers to discover new relationships and to increase their knowledge.

Annotated data is easier to integrate than unannotated data. It is far easier to select the correct data through the concepts they describe and context information, than selecting them using obscure cell coordinates and strings, which are often ambiguous and incomplete. To demonstrate the use of the RDF Record Table model for data integration, we have implemented support for this task as part of the Rosanne add-in for MS Excel.

In Rosanne, the user indicates which field is used to match records together (usually called the 'JOIN field' or sometimes the 'key') and which measurements they wish to select. Once this is done, the relevant records can be found automatically based on the annotated cells, and combined automatically using the information about the table structure.

The main challenge in integrating the annotated data is to combine the data stored in the different RDF Record Tables. SPARQL was the natural choice to perform this combination as it has the necessary functionality for searching, filtering and combining data expressed in RDFS/OWL.

In SQL, the relational equivalent to SPARQL, there is a standard functionality – the JOIN concept – which allows tables to be quickly combined. SPARQL does not have an equivalent concept, as SPARQL is based around the concept of triples, not of n-ary relations. It is standard in SPARQL to retrieve triples that share a subject, predicate or object, and in this way to combine the triples. However, integration of two records requires the integration not of two triples, but of two collections of triples. Most of these triples do not contain the common identifier, the JOIN field, on the basis of which the records are to be combined. The integration of tables is therefore more complex than the combination of isolated triples.

It was necessary for us to implement the JOIN functionality ourselves using building blocks from SPARQL, which was not a simple task. This is an important exercise for the Semantic Web, as it is becoming more and more common that tabular data is stored in RDF. We have developed a generic approach that is independent of the specific details of the tabular model, and therefore, which can work for both RDF Record Table and RDF Data Cube.

When integrating, there can be multiple records that have the same value for the JOIN field. For example, repeated measurements on the same sample. These multiple records must then be grouped together. For example, if we are joining records on the basis of the name "Jan", then the records "Jan, Wageningen, Tuesday" and "Jan, China, Tuesday" would be grouped together. To turn these records into one record, all fields except the JOIN field (which is by definition the same) must be aggregated.

Our approach follows these simple steps:

1. Select all relevant records (records containing the JOIN field)
2. Retrieve the desired information from the records
3. Group the records based on the JOIN values
4. Aggregate the other values
5. Structure all retrieved information into an integrated table
6. Retrieve the results

Steps 1 to 5 can be carried out within a single SPARQL integration query. This is a CONSTRUCT query, which creates a new RDF graph. The CONSTRUCT query nests three SELECT subqueries, which retrieve sets of variables from the existing RDF data. The innermost subquery selects the relevant records by looking for records containing an annotation that references the JOIN field (step 1). Optionally, the data to be included can be filtered in this step by using the SPARQL FILTER function, for example, we may only wish to integrate samples with a mass greater than 10g.

The second subquery selects the desired fields from the original records by looking for annotations with these fields (step 2) in the selected records, and groups the information based on the JOIN field (step 3). The outer subquery aggregates the data (step 4). Finally, the CONSTRUCT query forms the new records and creates the integrated table (step 5).

At this point, the integration is complete. However, the table is still in RDF, and is stored in the repository. To retrieve the results for recreating the two-dimensional table we use a second SPARQL SELECT query (step 6).

Note that, if wished, we could build the integrated table by simply collecting the data without aggregating them. The aggregation method, needed to construct a simplified, two-dimensional view, could then be specified when retrieving results, allowing different users to choose different views on the data. Either approach can be used depending on the situation.

It is possible that we may wish to join on more than one field. In the example above, we may not want the records about "Jan" to be merged if "Jan" is in different places. In that case, we need to identify the entities to join using both name and location.

The queries we have designed work with any number of tables. Naturally, there can be performance issues with large amounts of data.

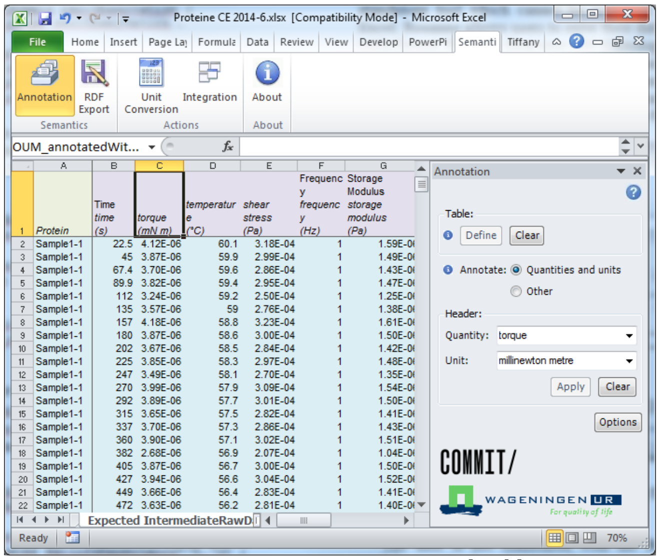
As previously mentioned, a common challenge in scientific data is that of handling *missing data*. When collecting records from different tables, we expect to find all available records in the result, even when data (in RDF Record Table values of `rec:containsContent`) is missing. By default, however, SPARQL expects all requested information asked for in the query to be present, otherwise no result will be returned. We solved this by use of OPTIONAL clauses in SPARQL. OPTIONAL allows a section of the requested data to be missing without preventing other results from being returned. A disadvantage of OPTIONAL is that it is slower, a known performance problem of this construct [30].

A specific case of missing data is when the JOIN field mentioned in a query is missing in the data. This is more likely to occur when there are multiple join fields – in our example the name "Jan" may always be filled in, but not always the location ("Wageningen" or "China"). In this situation, the way one merges different records depends on how he or she wishes to interpret the data. One option is that the user only wants to integrate records with the same name if the location is also the same, or if the location is missing in both. Alternatively, the user may wish to interpret the absence of location information as meaning 'any location', so that records with the same name will be integrated if both locations are the same, or if one or both locations are missing. Either choice can be catered for in the query. Our default is that records will only be matched if locations are identical. In addition, we implement the 'any location' interpretation by combining records with empty fields with all possible values of those fields, thus allowing integration with any value of that field. This is done within the integration query by means of an additional subquery.

We are currently investigating how we can support more complex queries, meeting the requirements that some users have specified for their practical cases. For example, in one situation, a scientist needs to 'integrate a measurement on a sample with the first record in time for that same sample, *after* the temperature of the sample has first exceeded 90 degrees Celsius'. For this purpose, we use the subquery facility in SPARQL to add further layers of nesting to the query. We have implemented such queries in a separate experimental tool and are now working on improving their performance and incorporating them into our main integration query. Supporting this type of queries will provide a significant benefit to researchers, as currently these integrations require a great deal of work and often specialized software or databases. How to provide a clear, intuitive user interface for such complex queries is an important issue.

All above-mentioned queries are independent of the precise tabular format. We have tested our basic integration approach developed for RDF Record Table on data in the RDF Data Cube format. The steps and the structure of the queries remain the same. The selection of the data fields is simply changed to use Data Cube syntax instead of RDF Record Table. The queries then work as designed.

For practical application of semantic integration functionality to be widely accepted, it has to be part of familiar, existing tools. Therefore we have incorporated it into our Excel add-in, Rosanne, extending the annotation functions presented before. Fig. 5 shows an example from food science. In this experiment, the researcher wishes to combine rheological measurements on protein samples with sample composition data. Without semantic support, this task would require her to find the relevant files somehow, then to copy and paste different data by hand, with plenty of scope for error. With semantically annotated tables, the necessary information is available to allow her to find the files via a search function (implemented in a demonstration tool but not yet incorporated into Rosanne). The tables have been annotated using OM and a domain ontology. The Integration Pane provides a list of all the concepts available in the files. The researcher selects 'Protein' as the identifier, and 'Storage Modulus' and 'Composition' as the variables of interest. Rosanne writes the RDF Record Table representations of the tables to a Sesame [31] repository, creates a SPARQL [27] CONSTRUCT query to find the relevant data, and generates the integrated table in the RDF Record Table format. A SPARQL SELECT query retrieves the data from the integrated table and writes it into a new



	A	B	C	D	E	F	G
1	Protein	Time time (s)	torque (mN m)	temperature (°C)	shear stress (Pa)	Frequency frequency (Hz)	Storage Modulus storage modulus (Pa)
2	Sample1-1	22.5	4.12E-06	60.1	3.18E-04	1	1.59E-0
3	Sample1-1	45	3.87E-06	59.9	2.99E-04	1	1.49E-0
4	Sample1-1	67.4	3.70E-06	59.6	2.86E-04	1	1.43E-0
5	Sample1-1	89.9	3.82E-06	59.4	2.95E-04	1	1.47E-0
6	Sample1-1	112	3.24E-06	59.2	2.50E-04	1	1.25E-0
7	Sample1-1	135	3.57E-06	59	2.76E-04	1	1.38E-0
8	Sample1-1	157	4.18E-06	58.8	3.23E-04	1	1.61E-0
9	Sample1-1	180	3.87E-06	58.6	3.00E-04	1	1.50E-0
10	Sample1-1	202	3.67E-06	58.5	2.84E-04	1	1.42E-0
11	Sample1-1	225	3.85E-06	58.3	2.97E-04	1	1.48E-0
12	Sample1-1	247	3.49E-06	58.1	2.70E-04	1	1.35E-0
13	Sample1-1	270	3.99E-06	57.9	3.09E-04	1	1.54E-0
14	Sample1-1	292	3.89E-06	57.7	3.01E-04	1	1.50E-0
15	Sample1-1	315	3.65E-06	57.5	2.82E-04	1	1.41E-0
16	Sample1-1	337	3.70E-06	57.3	2.86E-04	1	1.43E-0
17	Sample1-1	360	3.90E-06	57.1	3.02E-04	1	1.51E-0
18	Sample1-1	382	2.68E-06	56.9	2.07E-04	1	1.04E-0
19	Sample1-1	405	3.87E-06	56.7	3.00E-04	1	1.50E-0
20	Sample1-1	427	3.94E-06	56.6	3.04E-04	1	1.52E-0
21	Sample1-1	449	3.66E-06	56.4	2.83E-04	1	1.41E-0
22	Sample1-1	472	3.63E-06	56.2	2.81E-04	1	1.40E-0

Expected IntermediateRawD

Table: Define Clear

Annotate: [✓ Quantities and units] Other

Header:

Quantity: torque

Unit: millinewton metre

Apply Clear

Options

Figure 5: Rosanne using RDF Record Table.

Excel spreadsheet. The integrated table contains all the original annotations, and can itself again be integrated with other tables.

The process for the user is quite simple. She defines the integration they want with a series of simple dropdowns, and does not need to be aware of RDF Record Table, Sesame or SPARQL.

IX. EVALUATION OF ANNOTATION AND INTEGRATION ON INDUSTRIAL USE CASES

We tested annotation and integration via Rosanne on ten real-life use cases collected from four different academic research institutes, and the R&D departments of three commercial firms. These cases did not involve nested tables, but did include integration of more than two files, missing data and missing JOIN fields.

Regarding annotation, our key finding was that the provided data required some manual cleaning prior to annotation in Rosanne. Issues included JOIN variables being indirectly specified, for example, in the spreadsheet name, rather than being included in the table, tables being split over multiple locations in the spreadsheet, empty cells that were intended to be interpreted as including repetitions of previous cells, etc. Such issues can be addressed by adding data cleaning facilities to Rosanne, but are also related to compliance to good data notation by users. If information is completely missing or obscured, no tool will be able to recover it.

The integration function offered by Rosanne worked as desired for the majority of use cases once the data had been cleaned and annotated. The need for the more advanced integration queries as discussed in the previous section was confirmed by some of the other cases. These cases show how complex integration functionality, which would normally require researchers to turn to specialist solutions such as databases, can in future be offered in Excel in a simple, user-friendly manner.

One case involved a pivoted table, a table where the header contains data elements. As Rosanne does not yet support annotating the data elements within the header of a pivoted table, this table had to be unpivoted before it could be annotated. For the integration step itself, a wider range of data aggregation would be welcome. Some use cases, however, required advanced statistical analysis methods, such as regression, which fall outside the scope of aggregation. We are looking at a possible combination of Rosanne and a statistical package like R, rather than attempting to support this sort of analysis in SPARQL queries.

We conclude from these tests that RDF Record Table and our SPARQL approach for integration were successful in carrying out integrations on real-life data. While additional functionality is necessary to achieve the full results desired in some use cases, the core integration functionality was shown to be sound for a variety of data.

Due to the manual cleaning required, it was not possible to fully validate the performance and practicality of Rosanne in these use cases. Once the issues discovered in these use cases have been addressed, we will conduct a full validation of our approach.

X. CONCLUSION AND FUTURE WORK

We have proposed RDF Record Table as a way to model heterogeneous tabular data semantically. The model complements the RDF Data Cube vocabulary. RDF Data Cube offers the benefits of semantic modelling to domains such as statistics, with regular, standardized datasets, and provides good support for data visualization. RDF Record Table offers more flexibility in storing heterogeneous or incomplete data, and therefore extends the benefits of semantic modelling to the more complex situations for which RDF Data Cube is too restrictive. RDF Record Table addresses all four aspects identified in Section I as being essential for a good tabular model; the content can be annotated, the table structure is modelled, there is a link to the PROV model for provenance data, and it is flexible, allowing complex structures.

A first implementation of the RDF Record Table model as an extension of Microsoft Excel, called Rosanne, demonstrates that the format is capable of accurately representing tabular data, and can be applied by offering users simple choices from drop-downs, without the users needing to be aware of the RDF Record Table itself.

Rosanne also provides semi-automatic integration of datasets. SPARQL queries are used to integrate data from different RDF Record Tables. This integration approach is defined in a generic way, making it applicable to other RDF tabular models, such as RDF Data Cube. The user can specify their integration using simple drop-downs, and again does not need to be aware of the complexity of the model or the queries. This integration functionality has been evaluated in use cases from a number of research institutes and R&D organizations of multinationals in food production, cooperating in TI Food and Nutrition [32]. While a number of issues were identified that must be addressed to make Rosanne a practical tool for industry, the core integration principles were shown to be sound.

In Section I, we discussed the various problems that arise from how spreadsheet data is currently handled. RDF Record Table and its implementation in Excel provide a means to effectively tackle these problems. Ambiguity and incomprehensibility are addressed by linking data to defined concepts in shared vocabularies. The link between RDF Record Table and the PROV model allows the provenance of the data to be recorded. The annotations can be searched, making it easier to locate relevant data. The integration facility of Rosanne, built on top of the RDF Record Table model, enables data from different spreadsheets to be linked and combined together, assisting reuse. Finally, this support is available in the commonly used Excel tool, allowing researchers to incorporate good data bookkeeping into their research workflow, thus enabling good data documentation with minimal effort.

For full implementation of the RDF Record Table model, several issues must still be solved. While the model itself supports nesting, the Rosanne add-in does not. This support must be added, preferably by offering heuristics that assist the user. To handle the overhead of explicit annotations, RDF Record Table allows repeated information to be presented in cells that provide metadata for similar cells. However, methods for automatic (local) expansion and compression of datasets should be considered as well. In order to realize the benefits of both RDF Record Table and RDF Data Cube on the same data at different stages in the scientific or engineering process, mapping between the two formats is required. This necessitates support for implementing the constraints of Data Cube when converting Record Table to Data Cube.

As discussed, the integration functionality of the Rosanne add-in can be improved by allowing more complex queries, handling nested tables and offering a link to a statistical package for advanced data analysis.

Rosanne has not yet been optimized for performance on large datasets. This optimization will be a necessary step in producing an add-in that can be used in industry.

In addition to these issues on the annotation and processing of new data, the recovery of legacy data needs attention. There is a wealth of data stored in existing spreadsheets, which have, in general, an informal structure and no annotations. Current results for fully automatic annotation are still of insufficient quality [28], so more research is needed to find how to unlock this legacy data.

Semantic tables also offer the potential to support cleaning of the data, for example by defining allowed units and ranges for measurements so that errors can be detected and possibly (semi-)automatically corrected. This is an aspect that we will look at in the future.

A format for tabular data is of little use if it is not adopted by the community. We plan to submit RDF Record Table to the CSV on the Web Working Group [33] for consideration and inspiration in their work to provide better interoperability for tabular data.

Describing the content and structure of tabular data semantically makes it possible to easily find data even in disparate sources, to understand and clean the data and to combine it semi-automatically. This way, much richer datasets will be published in the future, so that others can fully understand them and build further on them.

ACKNOWLEDGMENT

This publication was supported by the Dutch national program COMMIT.

REFERENCES

[1] J. Top, H. Rijgersberg, and M. Wigham, "Semantically enriched spreadsheet tables in science and engineering," in Proc. Eighth International Conference on Advances in Semantic Processing (SEMAPRO), pp. 17-23, Rome, Italy, 2014.

[2] Y. L. Simmhan, B. Plale, and D. Gannon. "A survey of data provenance in e-science," ACM SIGMOD Record, 2005. doi:10.1145/1084805.1084812.

[3] A. Garcia, O. Giraldo, and J. Garcia, "Annotating Experimental Records Using Ontologies," Int. Conference on Biomedical Ontology, Buffalo, NY, USA, 2011. Available from: http://ceur-ws.org/Vol-833/paper12.pdf. Retrieved June, 2014.

[4] J. Gray, "Jim Gray on eScience: a transformed scientific method," in T. Hey, S. Tansley, K. Tolle (Eds.), The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, 2009, pp. xvii–xxxi.

[5] RDF Semantic Web Standards, W3C. Available from: http://www.w3.org/RDF/. Retrieved: May, 2015.

[6] R. Cyganiak, D. Reynolds, (eds). RDF Data Cube Vocabulary, W3C, 2012. Available from: http://www.w3.org/TR/vocab-data-cube/. Retrieved June, 2014.

[7] L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi, "RDF123: From spreadsheets to RDF," Lecture Notes in Computer Science, Vol. 5318 LNCS, 2008, pp. 451–466. doi:10.1007/978-3-540-88564-1-29.

[8] J. Cunha, J. Saraiva, and J. Visser, "From spreadsheets to relational databases and back," In Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation - PEPM "09 (p. 179), 2009.

[9] C. Bizer, and R. Cyganiak, "D2R Server – Publishing Relational Databases on the Semantic Web," World, p. 26, 2006.

[10] L. Lefort, J. Bobruk, A. Haller, K. Taylor, and A. Woolf, "A Linked Sensor Data Cube for a 100 year homogenised daily temperature dataset," In Proceedings of the 2012 5th International workshop on Semantic Sensor Networks – SSN 2012 (p. 1), 2012.

[11] S. Capadisli, S. Auer and A.-C. Ngonga Ngomo, "Linked SDMX Data," Semantic Web, 2013. doi:10.3233/SW-130123.

[12] Commercially Empowered Linked Open Data Ecosystems in Research project. Available from: http://code-research.eu/. Retrieved March, 2015.

[13] Tabels Project, Available from: http://idi.fundacionctic.org/tabels/. Retrieved March, 2015.

[14] TabLinker, 2012. Available from: http://www.data2semantics.org/2012/02/19/tablinker/. Retrieved June, 2014.

[15] RightField. Available from: https://www.sysmo-db.org/rightfield. Retrieved March, 2015.

[16] P. Rivera Salas, F. Maia Da Mota, M. Martin, S. Auer, K. Breitman, and M. A. Casanova, "Publishing Statistical Data on the Web," 2012 IEEE Sixth International Conference on Semantic Computing, Palermo, 2012. doi: 10.1109/ICSC.2012.16.

[17] OpenCube Toolkit. Available from: http://opencube-toolkit.eu/. Retrieved March, 2015.

[18] OpenCube Project. Available from: http://opencube-project.eu/ . Retrieved March, 2015.

[19] L. Lefort, H. Leroux, "Design and generation of Linked Clinical Data Cubes," First Internatianal Workshop on Semantic Statististics, Sydney, Australia, 2013.

[20] P. Groth, L. Moreau. (eds), PROV Overview, W3C, 2013. Available from: http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/. Retrieved June, 2014.

[21] M. Nilsson, A. Powell, P. Johnston, and A. Naeve. "Expressing Dublin Core metadata using the Resource Description Framework (RDF)," 2008. Available from: http://dublincore.org/documents/dc-rdf . Retrieved June, 2014.

[22] Matlab, The Language of Technical Computing. Available from: http://www.mathworks.nl/products/matlab/. Retrieved July, 2014.

[23] SPSS Statistics. Available from: http://en.wikipedia.org/wiki/SPSS. Retrieved July, 2014.

[24] The R Project for Statistical Computing. Available from: http://www.r-project.org/. Retrieved March, 2015.

[25] H. Rijgersberg, M. Wigham, and J. L. Top, "How semantics can improve engineering processes: A case of units of measure and quantities," Advanced Engineering Informatics, 25(2), 2010, pp. 276–287. doi:http://dx.doi.org/10.1016/j.aei.2010.07.008.

[26] R. Hodgson, P. J. Keller, J. Hodges, and J. Spivak, "QUDT - Quantities, Units, Dimensions and Data Types Ontologies," Available from: http://qudt.org/. Retrieved June, 2014.

[27] W3C, SPARQL Query Language for RDF. Available from: http://www.w3.org/TR/rdf-sparql-query/. Retrieved July, 2014.

[28] M. van Assem, H. Rijgersberg, M. Wigham, and J.L Top, "Converting and annotating quantitative data tables," The Semantic Web - ISWC 2010, vol. 6496/2010, 2010, pp. 16–31. doi:10.1007/978-3-642-17746-0_2.

[29] M.G. De Vos, W. R Van Hage, J. Ros, A.T. Schreiber, 2012. "Reconstructing Semantics of Scientific Models : a Case Study," In Proceedings of the OEDW workshop on Ontology engineering in a data driven world, EKAW 2012. Galway, Ireland.

[30] A. Loizou, R. Angles, and P. Groth. "On the Formulation of Performant SPARQL Queries," Web Semantics: Science, Services and Agents on the World Wide Web. Vol. 29, 2014 doi:10.1016/j.websem.2014.11.003.

[31] Sesame framework for RDF data. Available from: http://rdf4j.org/. Retrieved March, 2015.

[32] TI Food and Nutrition. Available from: http://www.tifn.nl. Retrieved July, 2014.

[33] CSV on the Web Working Group Charter, 2013. Available from: http://www.w3.org/2013/05/lcsv-charter.html. Retrieved June, 2014.