# RDF123: a mechanism to transform spreadsheets to RDF

Lushan Han, Tim Finin, Cynthia Parr, Joel Sachs and Anupam Joshi

University of Maryland, Baltimore County

August 26. 2007

We describe RDF123, a highly flexible open-source tool for transforming spreadsheet data to RDF. Existing spreadsheet-to-rdf tools typically map only to star-shaped RDF graphs, i.e. each spreadsheet row is an instance, with each column representing a property. RDF123, on the other hand, allows users to define mappings to arbitrary graphs, thus allowing much richer spreadsheet semantics to be expressed. Further, each row in the spreadsheet can be mapped with a different RDF scheme. Two interfaces are available. The first is a graphical interface that allows users to create their mapping in an intuitive manner. The second is a Web service that takes as input a URL to a Google spreadsheet or CSV file and an RDF123 map, and provides RDF as output.

## 1. Introduction

A significant amount of the world's data is maintained in spreadsheets. In this paper we present RDF123, a highly flexible open-source tool for transforming spreadsheet data to RDF. Our work is motivated by the fact that spreadsheets are easy to understand and use, offer intuitive interfaces, and have representational power adequate for most purposes. Moreover, online spreadsheets are becoming increasingly popular. These have the potential to boost the growth of the semantic web by providing well-formed and publicly shared data sources which could be directly maintained by end users but which can also be translated to RDF.

A drawback of spreadsheets is that their simplicity often results in data tables that do not follow the best practices of database design, such as attention to keys and normalization. Moreover, the liberty that people take with spreadsheets will sometimes require different rows to be translated with differing schemas. RDF123 addresses both of these issues.

In our approach, we borrow the idea from GRDDL [1] of placing a link in an online spreadsheet, which points to the corresponding map file. When an agent comes to the spreadsheet, it can follow the link, read the map file, apply the map file to the online spreadsheet and generate RDF data. The map is itself an RDF document. Our maps can permit a fairly complicated scheme to apply to a row, rather than just creating a single instance of a RDF/OWL class. We also adopt a general approach that allows different rows to use different schemas. For example, according to different values of a ‘sex’ column in a spreadsheet, we can generate a man instance or a woman instance conditionally.

We proceed as follows: Section 2 briefly contrasts our approach to other "spreadsheet to RDF" efforts, and also to GRDDL; Section 3 describes in detail the workings of the RDF123 translation; Section 4 gives a number of examples that illustrate usage, and is probably a good starting place for those eager to start defining mappings; Section 5 provides an architectural overview. We then offer brief concluding remarks, followed by references.

## 2. Related work

A number of programs have been developed to convert or export data from spreadsheets to RDF. The Maryland Mindswap lab developed two early ones: Excel2RDF [2] and the more flexible ConvertToRDF [3, 4]. Both these application assumed that an instance of a given class will be created for each row in the spreadsheet. The row's cells are used to populate the instance with property values and, typically, one provides an RDF node id for the instance. More sophisticated one (like D2R [5]) can specify mappings from a record set of a SQL query to instances of a RDF/OWL class.

One limitation of the above approaches is that the RDF schema used for one row or a group of rows is quite simple, usually having the shape of a star in which all property edges come out from a single center – the id resource. This works well for normalized database tables, but is not flexible enough for general purpose spreadsheets. Another limitation in the above approaches is that one fixed RDF schema is applied to all rows of a table.

An alternative approach is to create an XML representation of the spreadsheet, and then to use GRDDL, which has high flexibility, for the translation. However, the need to generate intermediate XML documents is a barrier. Consider an example. Suppose a restaurant maintains a published online spreadsheet showing the up-to-date menu items the restaurant is providing. The manager of the restaurant may want the menu items to not only be read by humans but also be read by agents and therefore available for semantic web queries. He also wants the machine-readable menu item data to be the most recent available. Since the data to be translated by GRDDL must be in XML (XHTML) format, the online spreadsheet translation has to include an additional step, that is, from data in spreadsheet to data in XML. This means that every time the restaurant manager modifies the menu items, he must take extra steps to push the spreadsheet data to XML and publish it; otherwise, an agent who reads the online XML will get stale data. Because the GRDDL standard stipulates that a GRDDL typed XML must has a link to a XSLT file, but does not have a standard about the link to the original data source, in our case, the spreadsheet, the agent that reads the GRDDL-typed XML file cannot fetch the latest data in the online spreadsheet automatically. If the agent could fetch directly from the spreadsheet, the intermediate XML file would be unnecessary. Another significant drawback of GRDDL translation is that the XSLT transform, which GRDDL relies on, is hard to create for users who are not XSLT specialists. The creation of an XSLT file, for many users, is no easier than writing a program. The mapping from tabular data to RDF should and can be done more intuitively than XSLT transformation.

# 3. Translation Design

## 3.1 Mapping Design

In order to define a more general mapping, we treat an RDF graph as a directed labeled graph, disregarding for the moment RDF schema concepts like classes and instances. Each vertex is either a resource or a literal. Each edge is an RDF triple. A resource with exactly the same label is treated as the same resource. A triple is also unique in a RDF graph. Every row of a spreadsheet will generate a row graph, and the RDF graph produced for the whole spreadsheet is the merge of all row graphs, eliminating duplicated resources and triples. The question is whether we can define a simple mapping which allows generated row graphs to take any shape, and also to vary significantly from one another.

To make the problem clear, we formally define the mapping from a spreadsheet to a RDF graph as the following:

$$G_{final} = \bigcup_{i=1}^{row\ count} G_i$$

$G_i$ is the row graph for ith row; $G_{final}$ is the ultimate graph

$$G_i = map(columns, i)$$

The *map* is a function that outputs a row graph for the ith row. The input to the *map* function includes "columns", an array of cell values of ith row, and the row number i. The computation of the function *map* only relies on the inputs of current row, not depending on the previous computation or future computation of the *map* on other rows. The row number i is a necessary input, usually used to distinguish this row from all other rows in a spreadsheet. A row RDF graph may have different number of vertices and/or edges from another row RDF graph. And in most cases, these row RDF graphs have a similar pattern, which means, for example, that some edges in different row graphs have the same label, or that the label of a vertex in different row graphs comes from the same column. If we overlap these row graphs by the similarity of vertices and edges, and then we look from the top, we end up with a graph which is a super graph of every row graph, with similar vertices/edges in different graphs converging on a single vertex/edge. (There could be alternative ways to converge the similar vertices/edges so that the super graph may not be unique.) This super graph is the basis of our mapping design, and we name it as the **map graph** or template.

In cases where the **map graph** needs to produce different labels for a converged vertex/edge, adapting to different row graphs, we put an expression instead of a static label in the place of the vertex/edge. The expression will be evaluated dynamically for each row and the result of the evaluation will be used as the ultimate label of vertices/edges in each row graph. The inputs of an expression are the same as the inputs of the *map* function – the array "columns" and the row number i. Expressions can use if-then-else logic and string manipulation operators to output a string as the evaluated static label for a vertex or edge. Since the **map graph** is a super graph of every row graph, for those vertices and edges which are in the **map graph** but are not in a row graph, the expressions will output empty strings. Our program will interpret an empty string as an instruction to not create the corresponding vertex/edge.

Note that if a vertex is not created, all incident edges on the vertex will not be created as well. The three elements – the **map graph**, the expressions in the map graph and the empty string equivalent to not creating – put together indeed compose one implementation of the *map* function. This *map* function can be used to generate all row graphs.

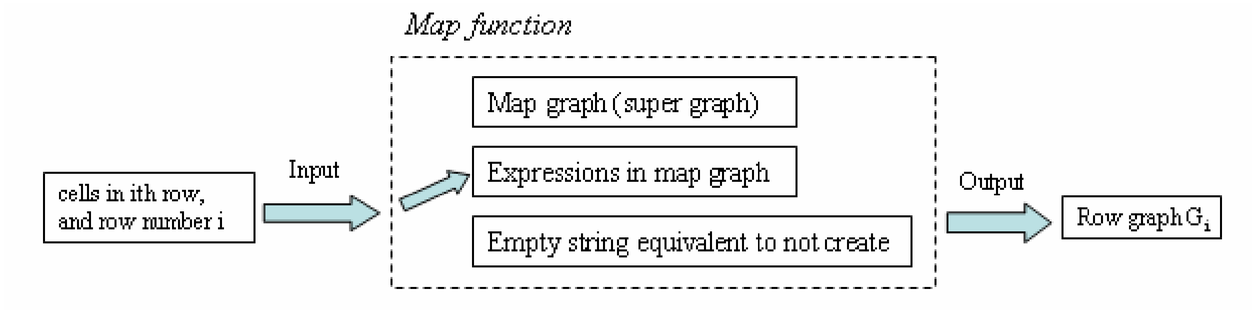


Figure 1. Our implementation of the map function

Our map function has high expressiveness, as we don't impose any constraints on every row graph. It can be arbitrary RDF graph. Because spreadsheets used by end users may not be normalized tables, arbitrary row graphs can maintain the expressiveness of spreadsheets.

On the other hand, a RDF123 mapping is more intuitive than an XSLT transformation because it can be expressed as a graph. In most cases, this **map graph** looks close to what the end user wants to construct from a spreadsheet. Moreover, the map graph can be serialized in an RDF document with RDF/XML, N3 and other common RDF syntaxes.

## 3.2 RDF123 Expression Design

Given a row in a spreadsheet and the row number, an expression in a converged vertex/edge is supposed to compute the final label of the vertex/edge for each row graph. Since expressions are part of the *map* function, their effectiveness affects the complexity of the map graph. In our current version, we build into it some boolean comparison operators and functions, a binary string operator (concatenation) '+' and some string manipulating functions, and an if-then-else ternary operator. The context-free grammar for the expression is defined as the followings:

```
E -> E+T | T                    -- Concatenation of strings

T -> @If(B;E;E) | S             -- Two sources of strings

B -> E=E | E!=E | @IsEmpty(E)   -- Boolean operations and functions

S -> '.*' | $[0..9]+ | prefix | -- Basic elements
     ? |                        -- Substitution for the base URI of
                                   the ultimate RDF document
     @Add(E,E) | @Sub(E,E) |    -- Add and Subtract
```

```
@Length(E) |                -- Length of a string
@IndexOf(E,E) |             -- The index within the first E of
                              the first occurrence of the second E
@Substr(E,E) |              -- A new string that is a substring of
                               the first E, starting at the index
                               given by the second E
@Substr(E,E,E)              -- A new string that is a substring of
                               the first E, starting at the index
                               given by the second E and ending at
                               the index given by the third E
```

@If(B;E;E) has a program logic like if-then-else, and it must output a string. The '.*' stands for any single quoted literal. $[0..9]+ is a regular expression to refer value in cells. For example, $1 refers to the first (with offset) cell in a row, $2 second, and etc. $0 has a special meaning in that it gives the current row number. If a literal is not enclosed by single quotation marks, it will be first checked whether it is a prefix. If the literal is a prefix name, its namespace URI is used instead; otherwise, the original literal is used in evaluation. Note that space character is illegal in the expression except for appearing in literals enclosed by single quotation marks. And expression is case-sensitive. Future extension of operators in expression design is in our plan.

## 3.3 Metadata for the map graph

Sometimes people need to describe the map graph with metadata such as creator, title, comments, type and so on. The metadata of a map graph can be inherited by the RDF documents generated by the map. These RDF documents usually have a common usage whereas the metadata of the map works like making stamps to them. Using a semantic web search engine, it is possible to locate all RDF documents marked with similar metadata. This benefits both data discovery and integration. For example, ecologists in different regions make observations about invasive species. They record observations in online spreadsheets, perhaps with different formats. They use map graphs to convert the spreadsheets to RDF documents for the purpose of data integration. If their map graphs share similar metadata, it is possible for a semantic web search engine to find their observation data regardless of their web addresses by collecting all RDF documents containing the inherited metadata.

Moreover, the '?' in the RDF123 expression can be used to make assertions about the ultimate RDF graph in the map graph. It can also allow users to specify RDF123 metadata of a spreadsheet, which will be covered in **Section 3.5**, in the map graph.

## 3.4 Serializing a map graph with RDF/XML syntax

Since a map graph is a template for producing row RDF graphs, it shares many characteristics with an ordinary RDF graph. It is beneficial to serialize the map graph with standard RDF encodings, like RDF/XML or N3, not only because it enables people who are familiar with RDF to edit the map graph manually but also because the map is structurally similar to the RDF data that is generated using it.. After we serialize the map graph to a file, we can publish the map file online to encourage reuse.

However, there are two subtleties about serializing the map graph that come with RDF123 expressions. First, in the case that expressions act as RDF resources in the serialized map graph, we have to forge a namespace for the expressions, as every resource is required to have a namespace. In RDF123, the forged namespace is "Ex:" which is not a prefix but a full namespace. Note that there is no ontology defined under this namespace. Aside from a namespace, the "Ex:" in a label also works as an identifier to its subsequent RDF123 expression so that the translation program can distinguish expressions from static labels in the map graph.

The second subtlety involves the W3C namespaces recommendation. Because it is quite likely that the ending character of an RDF expression is not in the required NCNameStartChar class (i.e., a letter or underscore), this will result in an empty local name when splitting the URI consisting of a RDF expression. Since a property with empty local name is not permitted in RDF/XML serialization, we can work around this by appending a character "_", which is, of course, in the NCNameStartChar class, to the end of a RDF expression. The optional ending character "_" has no effect on the interpretation of a RDF expression. Moreover, when an RDF123 expression acts as a URI in the serialized map graph, only characters legal in URI are allowed to populate the expression.

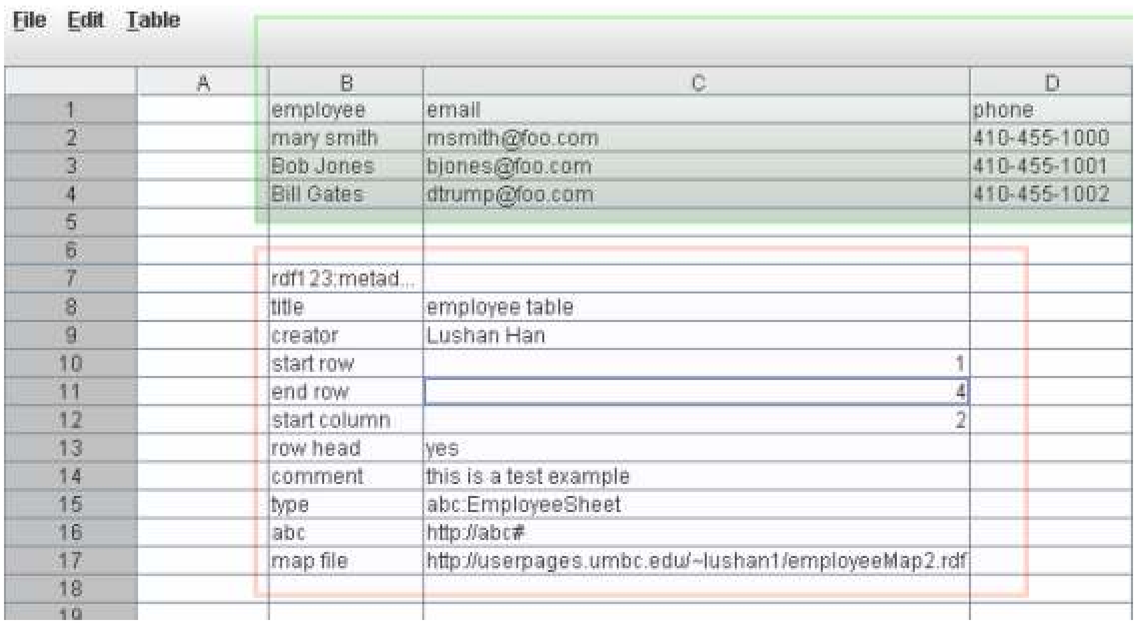
In some cases, it may be difficult to determine whether a vertex represents an RDF resource or a literal both in serializing the map graph and in populating the ultimate RDF graph. By the definition of RDF, the subject of a triple must be a resource. Accordingly, we can easily determine that vertices which have outgoing edges must be RDF resources. However, it is difficult to assess vertices which have no outgoing edges. One way to overcome this difficulty is to employ the range definition of the property represented by an incoming edge. This requires the translation program to preload the ontology which defines the property (However, except for the basic RDF ontology, currently the RDF123 application does not support preloading users' ontologies). However, not every property has a range definition. More complicatedly, the edge itself could be a RDF123 expression. We don't know what property it represents until the final label is evaluated from the expression. To work around this, RDF123 allows users to explicitly specify an XSD data types for a vertex in the map graph. If no XSD data type is explicitly specified to a vertex which has no outgoing edges, we choose the following strategy in serializing the map graph. If the label of the vertex is an RDF123 expression, we make it a literal unless we know the range definition of the property represented by its one incoming edge. If the label is not an RDF123 expression but a valid URI, we make it a resource; otherwise it is a literal. With this strategy, a type mistake could happen in the serialized map graph. However, the mistake will be corrected in populating the ultimate RDF graph when we get more concise information about the vertex.

## 3.5 RDF123 metadata of a spreadsheet

The target area that the map graph will be applied to must be a simple, regular table in a spreadsheet. However, besides a regular table, many spreadsheets have a few extra lines containing metadata about the spreadsheet and its columns. In order to let the transformation work properly, there must be a way to specify the exact spreadsheet region that will be transformed by the map graph. Also, people may have a need to generate some triples about specific metadata of the spreadsheet in the ultimate RDF graph as well. Moreover, it will be a good practice to embed the link to the online map file in the spreadsheet so that when the spreadsheet is discovered by a semantic web search engine, it can be transformed immediately to RDF data. Therefore, we also need a standard way to specify the link.

All these above metadata which are relevant to the RDF translation are put together into a contiguous and isolated tabular area with two columns and a header "rdf123:metadata" in a spreadsheet. When a program reads a spreadsheet, it will first scan all cells in the spreadsheet to see whether the tabular area with the header "rdf123:metadata" exists. If exist, the translation program will try to retrieve RDF123 metadata, use them in the translation process and store them in the output RDF graph. If not exist, a default setting is used, that is, the program will think the whole spreadsheet as a regular table with the first row being the header row.

In the RDF123 metadata area, people are allowed to tag the spreadsheet with a way similar to machine tag. The value of a tag could be a literal string or a RDF resource. If a tag with no namespace is used, the program will use default namespaces for some predefined tags. For example, the "comment" tag is interpreted as "rdfs:comment". Moreover, for users' convenience, RDF123 also predefines the prefixes of some popular namespaces, such as 'rdf', 'rdfs', 'owl', 'dc', 'foaf', 'sioc', 'vcard', and 'swrc'. Figure 2 shows an example with embedded metadata.



|  | A | B | C | D |
|---|---|---|---|---|
| 1 |  | employee | email | phone |
| 2 |  | mary smith | msmith@foo.com | 410-455-1000 |
| 3 |  | Bob Jones | bjones@foo.com | 410-455-1001 |
| 4 |  | Bill Gates | dtrump@foo.com | 410-455-1002 |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  | rdf123:metad... |  |  |
| 8 |  | title | employee table |  |
| 9 |  | creator | Lushan Han |  |
| 10 |  | start row | 1 |  |
| 11 |  | end row | 4 |  |
| 12 |  | start column | 2 |  |
| 13 |  | row head | yes |  |
| 14 |  | comment | this is a test example |  |
| 15 |  | type | abc:EmployeeSheet |  |
| 16 |  | abc | http://abc# |  |
| 17 |  | map file | http://userpages.umbc.edu/~lushan1/employeeMap2.rdf |  |
| 18 |  |  |  |  |

Figure 2. RDF123 uses a simple convention for embedding metadata for the translation using RDF123. This metadata can define properties of the RDF document produced (e.g., title), the range of the spreadsheet to be transformed, and the location of the RDF123 map

## 3.6 RDF123 map ontology

In order to make assertions about the RDF123 metadata of both the map graph and the spreadsheet in the output RDF graph, we need introduce a new vocabulary, which leads to the design of RDF123 map ontology. This ontology has three classes: Spreadsheet, ConvertedSpreadsheetInRDF, and Map. It has six properties: hasRowHead, startRow, endRow, startCol, hasSource, and hasMap. Figure 3 depicts the namespace with ovals representing classes and arcs represent properties. The informal relation graph shows the classes and properties introduced as well as the domain and range definition of properties. Properties hasStartRow and hasEndRow specify the start row and end row of the regular table area that will be translated to RDF. Property hasStartCol specifies the column offset of the area. This ontology is placed under the namespace: http://rdf123.umbc.edu/

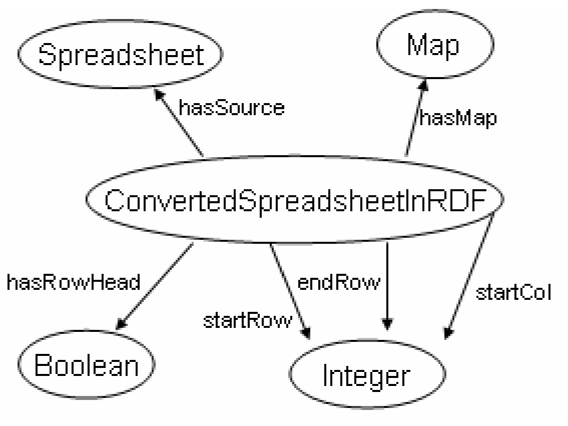


Figure 3. The RDF123 namespace has a simple set of classes and properties that represent the spreadsheet and support the transformation.

## 4. Examples

### 4.1 A simple RDF123 example

```
1    rdf123:metadata
2    title          Computing lab members
3    comment        8 June 2007
4    row head       yes
5    start row      7
6
7    NAME           EMAIL         OFFICE
8    Al Turing      amt@umbc.edu  ITE332
9    Don Knuth      dek@umbc.edu  ITE332
10   Marvin Minsky  mlm@umbc.edu  ITE442
...
```

Figure 4. A simple spreadsheet with contact information for members of a computing laboratory.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf123: <http://rdf123.org/> .


< > a rdf123:ConvertedSpreadsheetInRDF;
dc:title "Computing lab members;
rdfs:comment "8 June 2007".

[]  a foaf:Person;
foaf:name "Al Turing";
foaf:mbox "amt@umbc.edu";
foaf:offceNumber "ITE332".

[]  a foaf:Person;
foaf:name "Don Knuth"; ...
...
```

Figure 5. Desired RDF (shown in N3) for a simple spreadsheet with contact information for members of a computing laboratory.

Consider the spreadsheet shown in Figure 4, which shows contact information for the members of a lab. One reasonable choice for exporting this data as RDF might be as a set of foaf:Person instances using an additional property for the office number.  Figure 5 shows a sample of the RDF document that we might want to generate. The RDF representation of the map graph used in the translation is shown here encoded in N3

```
[] a foaf:Person ;
     foaf:name "Ex:$1" ;
     foaf:mbox "Ex:$2" ;
     foaf:officeNumber "Ex:$3".
```

The RDF header giving the namespace definitions is not shown for brevity. In this simple map, the anonymous instance is a template that will be created for each data row in the source spreadsheet. Within this template, the literal "Ex:$1" refers to the first cell in the row, "Ex:$2" to the second, etc. Because row RDF graphs have a homogeneous structure and coincide when superimposed, the map graph (super graph) shares the same structure as a row graph.

In this example, we don't assume that the values in the column "NAME" are unique. That is, name is not a key in the term of database terminology. Therefore, we create an anonymous instance (blank node) for each row. RDF123 supports blank nodes by assigning each blank node an internal id which is unique within the document. However, there are occasions where one would like to create instances with explicit ids instead of anonymous instances. Then, one can accomplish this by replacing the first line in the map graph with "*<Ex:'person'+$0> a foaf:Person;*". By the definition of RDF123 Expression, the variable '$0' outputs the row number, which is unique within a spreadsheet. The reason we prefix the id with the string 'person' is that the local name of a URI cannot start with a digit.

## 4.2 An example dealing with node id label

```
1    rdf123:metadata
2    title          Computing lab members
3    comment        8 June 2007
4    row head       yes
5    start row      7
6
7    NAME           EMAIL          OFFICE  SUPERVISOR
8    Al Turing      amt@umbc.edu   ITE332  Don Knuth
9    Don Knuth      dek@umbc.edu   ITE332  Grace Hopper
10   Marvin Minsky  mlm@umbc.edu   ITE442  Grace Hopper
...
```

Figure 6. A simple spreadsheet with contact information and supervisors for members of a computing laboratory. We want to assume that names are unique.

In Figure 6, we want to extend our simple example to include another column to represent a person's supervisor. This will force us to decide whether or not we should assume that two person entries with the same name refer to the same person. If we want to allow for the possibility that the two "Don Knuth" strings refer to different people, we might use the following in the map graph.

```
[]   a foaf:Person;
     foaf:name "Ex:$1";
     foaf:mbox "Ex:$2";
     foaf:officeNumber "Ex:$3";
```

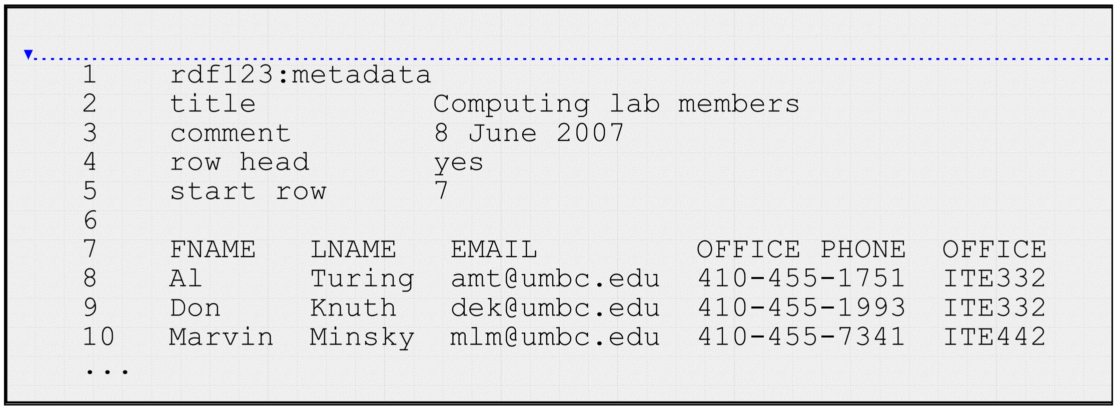
```
        foaf:supervisor
         [ a foaf:Person; foaf:name "Ex:$4". ].
```

For most simple spreadsheets, however, we might want to make a 'unique name assumption' for some fields, like the names of people and generate an RDF model with only one foaf:Person instance for "Don Knuth". We can accomplish this by using an id label with each foaf:Person instance.

```
   <Ex:$1>  a foaf:Person;
        foaf:name "Ex:$1";
        foaf:mailbox "Ex:$2";
        foaf:officeNumber "Ex:$3";
        foaf:supervisor
          [ <Ex:$4> a foaf:Person; foaf:name "Ex:$4". ]
```

### 4.3 Dealing with a non-normalized table

The example ion Figure 7 shows how a non-normalized table is translated to RDF. We slightly change our simple example from Figure 1 and include another column "OFFICE PHONE".



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 1 | rdf123:metadata |  |  |  |  |
| 2 | title | Computing lab members |  |  |  |
| 3 | comment | 8 June 2007 |  |  |  |
| 4 | row head | yes |  |  |  |
| 5 | start row | 7 |  |  |  |
| 6 |  |  |  |  |  |
| 7 | FNAME | LNAME | EMAIL | OFFICE PHONE | OFFICE |
| 8 | Al | Turing | amt@umbc.edu | 410-455-1751 | ITE332 |
| 9 | Don | Knuth | dek@umbc.edu | 410-455-1993 | ITE332 |
| 10 | Marvin | Minsky | mlm@umbc.edu | 410-455-7341 | ITE442 |
| ... |  |  |  |  |  |

Figure 7. Simple spreadsheet with contact information for members of a computing laboratory.

There is a function dependency (OFFICE PHONE -> OFFICE) in the table in Figure 6. Thus, the table is non-normalized. We want to generate additional instances for offices whose descriptions are assumed to be unique with the following map graph:

```
   []   a foaf:Person;
        foaf:name "Ex:$1+' '+$2";
        foaf:nick "Ex:$1";
        foaf:mbox "Ex:$3";
        foaf:officePhone "Ex:$4";
        foaf:officeNumber "Ex:$5";
        foaf:workAddress
          [<Ex:$5> an work:Office;
                   foaf:phone "Ex:$4";
                   work:number "Ex:$5". ].
```

RDF123 can be used to translate data in relational databases to RDF as long as we do table joins and output the denormalized table as a spreadsheet. The translation process may be more computationally expensive than directly translating from a database because there are many redundant data in rows. However, if the denormalized table is sorted, we can significantly reduce the total computation by storing the computation results of the immediately previous row and directly use the stored result if the input to a computational element is the same as the one in the previous row.

## 4.4 Applying different schemas to different rows

```
...

6
7    NAME           EMAIL         OFFICE  SEX     MATERNITY BONUS
8    Al Turing      amt@umbc.edu  ITE332  male    NULL
9    Jen Knuth      jen@umbc.edu  ITE332  female  $100
10   Marvin Minsky  mlm@umbc.edu  ITE442  male    NULL
...
```

Figure 8. We add a column for the employee's sex and the value of a special maternity bonus.

In the last example in Figure 8, we include two new columns. One is "SEX" and the other is "MATERNITY BONUS". We want to create different instances, a man or a woman instance, depending on the value in the column "SEX". Additionally, a woman instance has a "MATERNITY BONUS" property, which is missed for a man instance. The expected row graphs for row eight and row nine are shown in Figure 9.

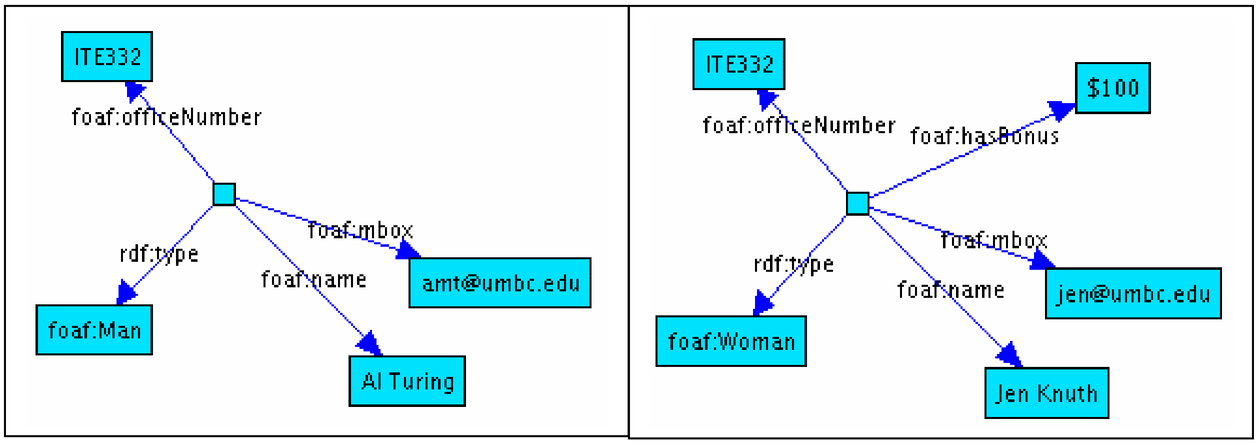


Figure 9. The RDF graph on the left is generated for Mr. Turning of row eight and the one on the right for Ms. Knuth of row nine. Note that their types differ and Mr. Turning has no bonus property at all.

Depending on different values of column SEX, the schema adopted for each row varies. If the value of the field SEX is ‘male’, a man instance is created for that row. If the value is ‘female’, a woman instance is created instead and the instance has an additional property “foaf:hasBonus”.

The map graph (super graph) is shown in Figure 10:

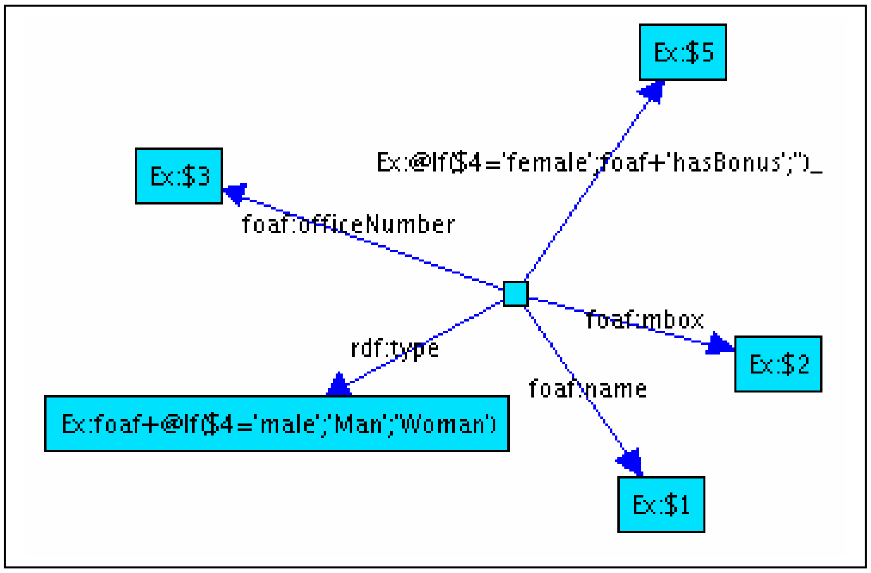

Ex:$5
Ex:$3
Ex:@If($4='female';foaf+'hasBonus';'')_
foaf:officeNumber
foaf:mbox
Ex:$2
rdf:type
foaf:name
Ex:foaf+@If($4='male';'Man';'Woman')
Ex:$1

Figure 10. The RDF map for the spreadsheet in figure 8.

Every vertex and edge in both row graph 8 and row graph 9 is included in the map graph, with similar vertexes/edges converging on a single vertex/edge. Instead of static labels, we use expressions for the converged vertexes and edges. In this example, one convergence happens. The two vertices “foaf:Man” and “foaf:Woman” in different graphs converge on a single vertex with the expression “Ex:foaf+@If($4='male';'Man';'Woman')”. This expression means that if the *sex* field have a value ‘male’, it will create a “foaf:Man”; otherwise a “foaf:Woman”. We also need notice that “foaf:hasBonus” property should not be created for row graph 8. We accomplish this by using an expression “Ex:@If($4='female';foaf+'hasBonus';'')_” in the edge that represents this property in the map graph. If the SEX field has a value “female”, the edge will be evaluated to “foaf:hasBonus”, otherwise an empty string. By the design of our map function, the empty string means not to create the edge, which is what we require for the row graph 8.

Depending on different ways we converge vertexes/edges, the map graph may not be unique. The following gives two alternative map graph serialized in N3.

```
# here is one way to encode the map
[] a <Ex:foaf+@If($4='male';'Man';'Woman')> ;
     foaf:hasBonus "Ex:@If($4='female';$5;'')" ;
     foaf:mbox "Ex:$2" ;
     foaf:name "Ex:$1" ;
     foaf:officeNumber "Ex:$3" .
```

```
# here is another way to encode the map
[] a <Ex:foaf+@If($4='male';'Man';'Woman')> ;
     <Ex:@If($4='female';foaf+'hasBonus';'')_> "Ex:$5" ;
     foaf:mbox "Ex:$2" ;
     foaf:name "Ex:$1" ;
     foaf:officeNumber "Ex:$3" .
```

Rather than viewing the map graph as a single schema, it would be better to view it as a combined schema. The translation from a spreadsheet to an RDF graph is the process of applying the map graph to each row to produce row graphs and then merging all of them to get the ultimate RDF graph.

## 5. RDF123 Architecture

As shown in Figure 11, RDF123 consists of two components, the RDF123 application and the RDF123 web service.

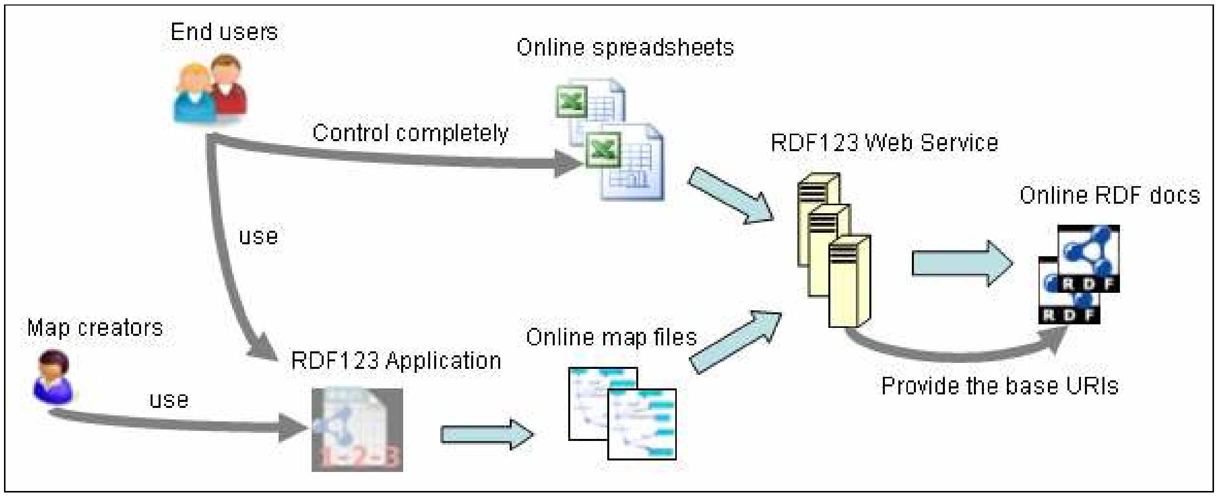


Figure 11. RDF123 Architecture Overview.

### 5.1 RDF123 application

The main purpose of the RDF123 application is to give users an interactive and easy-to-use graphical interface for creating the map graph and outputting the map graph in RDF syntax like RDF/XML and N3. As an RDF document, the serialized map graph can be manually edited by people and published online to encourage reuse and extensibility. The RDF123 application also supports a full work cycle of translating a spreadsheet to RDF, including importing CSV file into a graphical spreadsheet editor and translating the spreadsheet to RDF by applying the map graph.

RDF123 application is composed of three internal frames. The first frame "prefix definition" works as a prefix library. Users can store their (namespace, prefix) pairs in a prefix list. name-spaces are usually long and forgettable URLs, which are very hard to operate for many users. The good thing with the "prefix definition" list is that once a prefix definition is stored, users never need deal with the full namespace in the whole translation process. Use the prefix wher-ever a namespace is required. The second frame is a spreadsheet editor, which enable users to open a CSV file, edit the file in a similar way like Excel, and save the file. The third frame is an interactive graph editor that allows user to create and remove a vertex/edge, drag a vertex, and change properties of a vertex/edge. Users create their own map graphs with this graph editor. Labels on the nodes in the map graph can be used to create blank nodes or labeled nodes, attach an XSD datatype, and create RDF123 expressions. People can use the prefix wherever a name-space is required. The map graph can be saved to a local file which stores the positions and col-ors of the vertices/edges in the graph for the purpose of future modification, or it can be serialized to an RDF document with RDF/XML or N3 syntax. After the map graph is created, users can translate the spreadsheet to RDF by invoking a menu command.

The ultimate goal of the graph editor is to make creating a map graph easily handled by a user who knows little or nothing about RDF. Currently, from a user's perspective, creating a map graph is no harder than drawing a relation diagram for the fields in his spreadsheet, except for the namespace selection. It would be very convenient if a user could just type an English word for a class or a property and the program can automatically choose the most proper namespace which defines the class or property. In the future, we may implement this feature by utilizing ranks of terms on the semantic web, by inferring the map's domain through the column headers, and guessing the best choice with Bayesian Belief Networks based on namespaces already used and prior experience.

## 5.2 RDF123 Web Service

The RDF123 web service aims to provide a public service that translates online spreadsheets to RDF and also works as the host of the URIs of the RDF documents that come from online spreadsheets. The service is built on the HTTP Get protocol. The service URL is http://rdf123.umbc.edu/server/ and it takes three basic parameters: 'src', 'map' and 'out'. If a spreadsheet has an embedded link to its online map file, we just need to specify the URL of the spreadsheet with the 'src' parameter; otherwise, we also need give the location of the map file with the 'map' parameter. The parameter 'out' is used to specify the output syntax. An additional parameter "gid" is used to specify the sheet id within a spreadsheet that has multiple sheets. Examples are available at http://rdf123.umbc.edu/examples/

The RDF123 web service should not be a centralized service. We recommend that people host the RDF123 web service on their own websites not only because of a scalability concern but also because the website revealed from the URI of an online RDF document can sometimes be used as evidence of trust.

A function of the RDF123 web service that should not be overlooked is that it provides the base URIs for the RDF documents which are translated from online spreadsheets. Any RDF document must have a unique URI so that people and machines can refer to it by the URI. Once the URI provided by the RDF123 web service is invoked, the web service will read the online spreadsheet and the map file given as the parameters in the URI, apply the map to the spreadsheet and output the RDF data dynamically. In this architecture, end users have the complete control over the content of the online RDF document translated from their spreadsheets. Moreover, they just need to deal with the data in their spreadsheets, which is easy, and the content of online RDF documents will change correspondingly.

## 5.3 RDF123 Map Layer

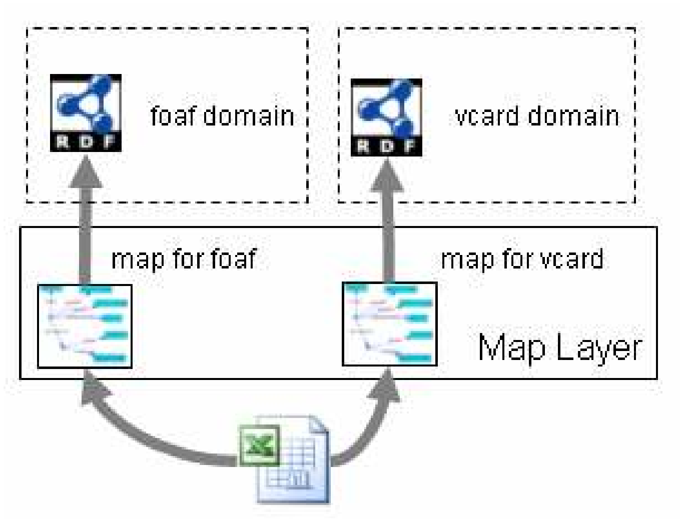


Figure 12. Separating the spreadsheet data and the maps used to convert them to RDF makes it easy to generate different RDF encodings from the same spreadsheet, encouraging data interoperability and reuse.

Adding a lightweight map layer between the original data source in spreadsheets and ultimate RDF data can probably smooth the web data integration process because people can choose their own preferred ontology to create the map and therefore make the data available in their own domain. For example, biologists may have different ontologies about how to describe an observation entity. If they cannot agree on which ontology to use, there will be an ontology mapping issue to solve before they can integrate their data. RDF123 provides an alternative way to integrate their data. Rather than putting the data in an RDF document written with a fixed ontology, and publishing the URI of the RDF document, it would be better to keep the data in a spreadsheet and publish the URI of the converted spreadsheet provided by the RDF123 web service. From a machine's perspective, the RDF data retrieved from the two approaches is identical. However, the second approach enables us to keep data in its original format. There are three benefits. One is that the same data can be available in different domains just by associating it with different map files. The second is that when the ontology evolves and changes, in order to make the data adapt to the change, what we need to do is to modify the map file instead of regenerating the hard coded RDF document. The third is that the map layer can split the work of generating RDF. Spreadsheets are good for end users to manage their own data, and the map file could be created with the help of RDF specialists.

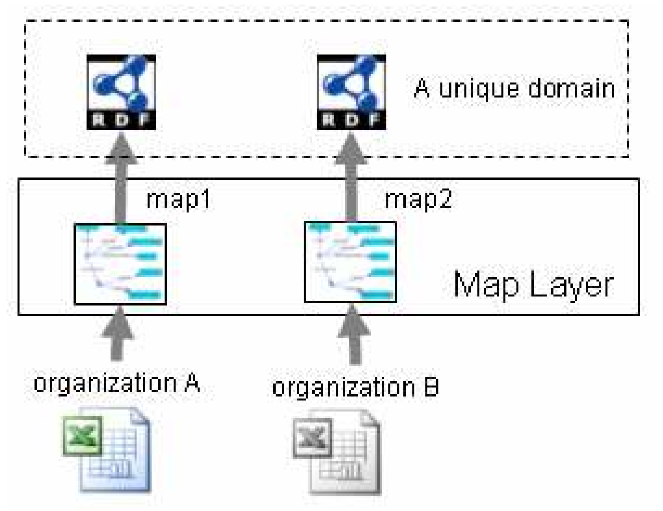


Figure 13. Separating the data and maps also enables two different organizations to develop maps for spreadsheet data in their own unique formats but mapped to the same RDF ontology.

In other cases, the map layer can also play a role in retrieving the same kind of data from heterogeneous spreadsheets created by different organizations, and making them available in a unique domain. For example, researchers who do statistics sometimes need to collect data from different sources, many of which are in the form of spreadsheets. However, these spreadsheets, although describing almost the same kind of thing, usually have different formats and duplicated data. In order to conduct statistical analyses, researchers must do heavy pre-processing work to ensure the data have the same format so they can be stored in a single spreadsheet. With the help of RDF123, a researcher can accomplish this merging task easily by defining a map for each spreadsheet and translating all of them to RDF using a unique ontology. Then, she can import these RDF documents to a triple store and use a SPARQL query to output data in tabular format which can be accepted by a statistical analysis program.

## 5.4 Publishing RDF

How can we publish the RDF data converted from spreadsheets? One way is to publish the URI provided by the RDF123 web service in the same way we publish a physical RDF document: submitting the URI to a semantic web search engine such as Swoogle [6], or creating a web page embedding a href link to the URI.

Could we use traditional search engines like Google to help us find possible spreadsheets that could be converted to RDF? Google already supports searching on CSV and Excel files on the whole web. It has indexed 1,350,000 CSV files and 14,700,000 XLS files. We have already discussed how to embed RDF123 metadata in spreadsheets. If we use Google to query for keywords that are particular to RDF123 metadata like “rdf123:metadata”, “map file” or specific value of a tag like ‘type’ as well as the file type constraint, we are able to harvest all spreadsheets of a particular type that can be converted by RDF123. Thus, there could be a very simple way for end users to publish their own data. Many RDF123 spreadsheet templates about different subjects can be distributed among end users. End users can fill in their own data and publish the instantiated spreadsheet. Once Google indexes these documents, a semantic web search engine is able to convert them to RDF without the involvement of end users.

The RDF123 web service has the potential to make a huge amount of data in the form of heterogeneous spreadsheets available to the semantic web or other applications which require the data in a homogenous format.

## 6. Conclusions

Spreadsheets are widely used to store and maintain simple data collections. The structural simplicity of the data stored in many spreadsheets makes it relatively easy to export the data into an RDF format. We have described RDF123 as an application designed to make it easy for end users to develop a map between their spreadsheet data and RDF and to use this map to generate RDF data serialized as either XML or N3. The RDF123 web service allows agents to translate spreadsheets as they are encountered, ensuring that data is always current, and obviating the need for maintaining a separate RDF repository on-line.

## 7. References

[1] Gleaning Resource Descriptions from Dialects of Languages (GRDDL), Dr. Hazael-Massieux and D. Connolly, W3C Coordination Group Note NOTE-grddl-20040413, World Wide Web Consortium, April, 2004.

[2] http://www.mindswap.org/%7Erreck/excel2rdf.shtml

[3] http://www.mindswap.org/~mhgrove/ConvertToRDF/

[4] J. Golbeck, M. Grove, B. Parsia, A. Kalyanpur and J. Hendler, New Tools for the Semantic Web, Proceedings of EKAW, pp 392-400, Springer, 2002.

[5] C. Bizer, D2R MAP – A Database to RDF Mapping Language, Proceedings of the 12th International World Wide Web, 2003.

[6] http://swoogle.umbc.edu