RDF123: from Spreadsheets to RDF *

Lushan Han[1], Tim Finin[1], Cynthia Parr[2], Joel Sachs[1], and Anupam Joshi[1]

[1] University of Maryland, Baltimore County, USA
{lushan1,finin,jsachs,joshi}@cs.umbc.edu
[2] University of Maryland, College Park, USA
csparr@umd.edu

Abstract. We describe RDF123, a highly flexible open-source tool for translating 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 fairly different RDF scheme. Two interfaces are available. The first is a graphical application 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.

Key words: RDF, Spreadsheets, Web services, Data interoperability

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 translating 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 many common purposes. Moreover, online spreadsheets are increasingly popular and have the potential to boost the growth of the Semantic Web by providing well-formed and publicly shared data sources that can be directly maintained by users and automatically translated into 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, let alone the richer features enabled by knowledge bases. Moreover, the liberty that people take with spreadsheets will sometimes require different rows to be translated with different schemata. RDF123 addresses both of these issues. RDF123's translation from a spreadsheet to an RDF graph is driven by a map which permits a rich schema to apply to a row, rather than just creating a single instance of a RDF/OWL class. We also adopt a general

* Apologies to Jonathan Sachs and Mitch Kapor. Partial support for this research was provided by the National Science Foundation through awards ITR-IIS-0325172 and NSF-ITR-IDM-0219649.

approach that allows different rows to use fairly different schemata. For example, depending on the value in the spreadsheet's column labeled 'sex', we generate a 'Man' instance or a 'Woman' instance.

In our approach, we borrow the idea from GRDDL [1] of placing a link in an online spreadsheet referencing the RDF123 translation map, which is itself an RDF document, specifying the desired translation. When an agent comes to the spreadsheet, it follows the link, reads the map file, applies it to the spreadsheet and thus generates RDF data. Moreover, RDF123's Web service also allows users to apply map files to other users' online spreadsheets and generate their customized RDF data.

The remainder of the paper proceeds as follows. Section 2 briefly contrasts our approach to other systems that map spreadsheets or database tables into RDF graphs and also to GRDDL. Section 3 describes in detail the workings of the RDF123 translation. Section 4 describes the approach to representing a *map graph* as an RDF document. Section 5 explains how to specify metadata in RDF123. Section 6 provides an architectural overview of the system. We conclude the paper with some brief remarks and identify issues for future research.

2 Related Work

Several programs have been developed to convert or export data from spreadsheets to RDF. The Maryland Mindswap Lab developed two early systems: Excel2RDF [2] and the more flexible ConvertToRDF [3, 4]. Both these application assumed that an instance of a given class should 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 itself.

The Babel system that is part of the MIT Simile suite of tools [5] can extract data from excel spreadsheets and from tab-delimited tabular data and render it in JSON and eventually RDF.

The TopBraid Composer [6] Semantic Web development system can extract class and instances information from spreadsheets and these can be further manipulated and transformed using additional tools in the suite.

One limitation of the approaches described above 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.

A problem that is very similar is generating RDF data from a relational database. A more sophisticated translation system, such as like D2R [7], can specify mappings from rows in the result set of a SQL query to instances of a RDF/OWL class. The approach uses D2R MAP [8], which is a declarative language to define mappings between relational database schemata and OWL/RDFS ontologies. Using the SQL query language to define mappings yields a system with considerable representational power and flexibility, but requires that its

users have considerable familiarity with relational databases and SQL. Moreover, its applicability is limited to databases, which are, for the most part, developed and used by IT professionals rather than IT users.

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 software 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, transforming the data in the 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. 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 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 and 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 result of merging all of the row graphs, eliminating duplicated resources and triples as necessary. We would like to define simple mappings that allow the row graphs to take any shape, and also to vary significantly from one another.

We formally define the mapping from a spreadsheet to a RDF graph as the following, where G_{i} is the row graph for the i^{th} row and G_{final} is the ultimate graph.

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

$$G_i = map\,(rowCells[\,], i) \ . \tag{2}$$

The *map* function produces a row graph for the i^{th} row given an array of its cell values and the row number i. The computation of the function *map* only relies on the inputs of current row and not on the previous computation or future

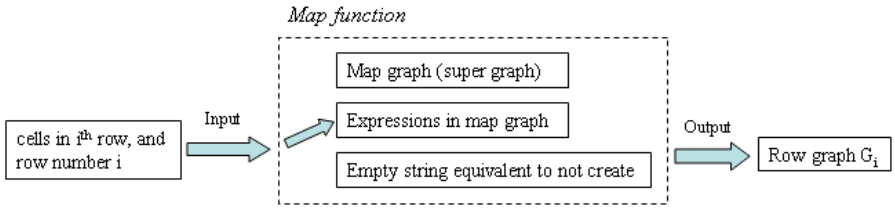

Fig. 1. Three elements in our implementation of the *map* function

computation of the *map* on other rows. The row number i is a required input, used to generate unique IDs or labels spanning the whole RDF graph. Two row graphs may differ in their number of vertices and/or edges, but they will typically have a similar pattern. For example, some edges in different row graphs will have the same label, or the labels of vertices in different row graphs come from the same column. If we overlap these row graphs by unifying vertices and edges, and then we look from the top, we end up with a graph that is a super graph of every row graph, with similar vertices/edges in different graphs converging on a single vertex/edge. (There can be other ways to merge the similar vertices/edges so that the super graph may not be unique.) This super graph is the basis of our mapping design, which we call the *map graph* or template.

When the *map graph* should produce different labels for a converged vertex or edge in different row graphs, an expression is used for the vertex or edge rather than a static label. The expression is evaluated for each row and the result 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 rowCells [ ] and the row number i. Expressions can use if-then-else sub-expressions and string manipulation operators to compute a string as the final 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 absent from a row graph, the expressions will output empty strings, which signal that no vertex or edge should be created. Note that if a vertex is not created, no incident edges are created as well.

The three elements, the *map graph*, *map graph* expressions, and the convention that empty strings generate no vertices or edges, characterize the *map* function and render it able to generate all row graphs, as shown in Figure 1.

The *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 is expressed as a graph and can be visualized and authored with RDF123 graphical application. Typically this *map graph* resembles a diagram of entities and their relationships that captures what users have interpreted from a spreadsheet. As you would expect, the *map*

graph can be serialized in an RDF document with RDF/XML, N3 and other common RDF serializations.

3.2 RDF123 Expression Design

The role of RDF123 expression is to compute the final label for a converged vertex/edge in a *map graph*, depending on the input of cell values of a row and the row number. The expressiveness and simplicity of RDF123 expression are both important because they determine the complexity of the *map graph*. RDF123 expression is defined by a context-free grammar and is able to do branch, arithmetic and string processing operations. All these operations, including branch, are themselves expressions that can be recursively embedded in other expressions. Expressions strings used as input to a parent expressions, with the value of the outermost expression serving as a final label for a vertex or edge, provided it is not the null string. While string concatenation and equality use an infix notation, other operations employ a functional notation. For example, branch expression is defined as $@If(arg1; arg2; arg3)$ and addition as $@Add(arg1, arg2)$.

Since RDF123 is implemented in Java, string manipulating methods in the java.lang.String class are easily exposed as RDF123 expressions and common $@Length$, $@IndexOf$ and $@Substr$ methods are available. To maintain a conceptually simple model, there are no other data types such as number or boolean in RDF123 expressions. However, strings are coerced to the appropriate data type when the semantics of the operation and the operand require other types. Exceptions may happen during conversion, which leads to the two running modes of RDF123 program. One mode is to produce as many triples as possible. In this mode, any exception will result in an empty output string which means not creating the vertex or edge but the whole program will continue running for other vertexes and edges. The other mode simply terminates the whole program and returns an error message. In order to have a neat display, RDF123 expression also allows using prefix instead of writing whole namespace.

Not every converged vertex/edge has a label that must be computed or transformed; some are simple static labels. To distinguish dynamic and static labels for converged vertexes/edges, we introduce a pseudo namespace '*Ex*'. If a label begins with '*Ex:*', it simply means that the following string is a RDF123 expression; otherwise a static label. The use of this pseudo namespace makes the *map graph* have the form of an RDF graph and enables it to be serialized in many forms, such as RDF/XML and N3. (See Figure 2 for an example.) The normal RDF semantics does not apply to RDF123 map graphs, of course. Luckily, they are easily recognizable via their metadata annotations and should ultimately pose no more problems that document templates and samples do for human readers.

3.3 Determining the type of a converged vertex

The role of an RDF123 expression is to produce a final label for a converged vertex or edge. It is more like a process of data extraction and transformation.

$$Ex: @If(\$2 != \$3; foaf+'knows';'')$$

Fig. 2. The terms $2 and $3 in this RDF123 expression denote the cell values in columns 2 and 3. The expression computes a dynamic edge: if the two cell values are not equal we generate 'foaf:knows' for the converged edge; otherwise nothing is generated. $[i] denotes cell values in column i for current row when $i > 0$ while $[0] gives the current row number.

However, we also need know the RDF element type for the converged vertex or edge before we can output the data as RDF. For edges, it is very simple because they are always *rdf:Property*. But for vertices, it is a little bit tricky because the potential type could be one of several data types (e.g., *rdf:Resource*, *rdf:Literal*, XML data types) or even composite data types like RDF container, collection or object group. We can divide the possibilities into two general cases. For those vertices which have outgoing edges, we can conclude that they should be of type *rdf:Resource*. When it comes to those leaf vertices, we allow users to explicitly append a vertex type at the end of the static label or RDF123 expression. For example, *Ex:$1^^integer*. When lacking an explicit data type, we take the following heuristic: if the final label is a valid URI, we make it a *rdf:Resource* otherwise a *rdf:Literal*.

When the specified vertex type is a composite data type, like *rdf:Bag*, we require that the vertex label must follow a certain syntax, such as Prolog list, so that a parser can understand it and put it to the corresponding RDF data. The atomic elements in the list can also have vertex type appended so that it is possible to generate a bag of *rdf:Resource* or *rdf:Literal* dependently. RDF123 expressions can provide some basic functions to help users format their data to syntactically correct list. Composite vertex types are not supported in current version of RDF123.

An object group is also a composite data type, but is different from RDF container or collection with respect to how the data is transformed to RDF. For each element in the object group, we create a separate assertion instead of one assertion to the whole set. For example, consider a spreadsheet for school classes. A class can have one instructor and multiple students, which are stored in only two columns 'instructor' and 'students'. We would like to generate a foaf:knows assertion from the instructor to every students respectively. In this case, we can use an object group vertex type for 'students'.

RDF123 also supports blank nodes. To create a blank node, just leave the label of a vertex completely empty. Be careful that 'Ex:' has a completely different semantic because it is interpreted as not creating the vertex. Actually all blank nodes have internal IDs in a physical RDF storage model. In RDF123, the row number i is used to generate a unique internal ID for a blank node.

3.4 A Simple Translation Example

People like spreadsheets because they provide a convenient way to capture the similarity of data, group and store similar data together in a succinct, informal schema. This schema may be easily criticized by database specialists because it is hard to store and query. However, it has the advantage of being intuitive. RDF123 *map graph* is a template that copies the intuitive schema from a spreadsheet and allows subtleties and dissimilarities within similarity to be expressed with RDF123 expressions. Generally speaking, a vertex in a *map graph* can often find its corresponding column in a spreadsheet and an edge simply comes from an interpreted semantic relation between two columns. RDF123 expressions and vertex type play a role of refining data and transform data to RDF, a machine-understandable schema, from an intuitive but informal schema. Let's see one example.

Suppose that the UMBC CS department has a research club that includes faculty and students. Faculty are required to pay a small amount of money for monthly coffee dues, but students are not. Table. 1 shows a spreadsheet and Figure 3 the corresponding *map graph*.

Table 1. A simple spreadsheet for the members of a research club.

Name	**Email**	**Office**	**Faculty**	**Coffee Due**	**Advisor**
Tim Finin	finin@umbc.edu	ITE329	Yes	$10	
Lushan Han	lushan@umbc.edu	ITE377	No		Tim Finin
Wenjia Li	wenjia@umbc.edu	ITE377	No		Anupam Joshi

The *map graph* follows the intuitive schema of the spreadsheet but expresses some subtleties using RDF123 expression. The expression *Ex:foo+@If($4='Yes'; 'Professor'; 'Student')* will produce a resource 'foo:Professor' for the rows having 'Yes' in FACULTY column and 'foo:Student' for the others (where 'foo' is a hypothetical namespace). The expression *Ex:$5^^decimal* specifies a vertex type 'decimal' because the coffee due has monetary value. For the rows representing students, *Ex:$5^^decimal* will output an empty string and therefore the corresponding vertex along with the incident edge 'foo:hasCoffeeDue' will not be created for students. Besides information in columns, we add a general assertion that all generated instances are members of a club instance named 'UMBC CS Friday Afternoon Research Club'. Because no namespace is specified for the local name 'UMBC CS Friday Afternoon Research Club', the club instance will, by default, have the online document base URI as its namespace. Although the club instance is created for each row graph, they share the same resource URI. Therefore, after doing a union of all row graphs, only one club instance remains.

This spreadsheet example implicitly uses the 'unique name assumption' because a person name's is used as node id of instances, such as *Ex:$1* and *Ex:$6*.

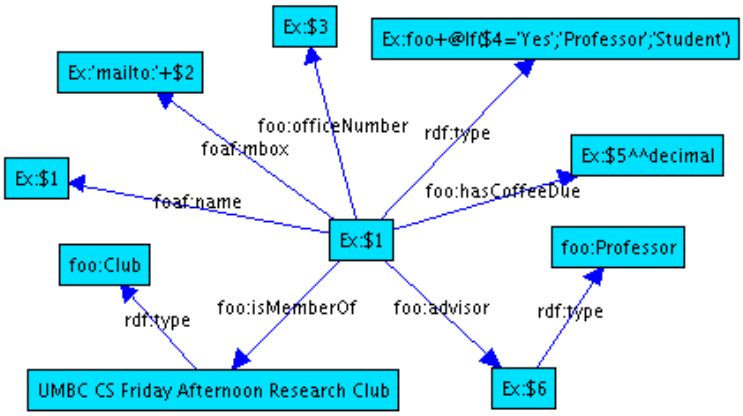

Fig. 3. The corresponding *map graph* made with RDF123 graphical application.

Typically, we would like to have a unique resource URI for referencing the same instance appearing in different places of a spreadsheet. Doing so allows different assertions about an instance to come together. For example, the second row of the spreadsheet tells us that Lushan Han's adviser is Tim Finin. And the person Tim Finin has an email address 'finin@umbc.edu', which is actually obtained from the first row. In most spreadsheets, the unique name assumption is implied because their authors would certainly hope to see that potential readers can disambiguate person names. If two people share the same name, the author might introduce the middle name or use another notation to differentiate them. When such a unique name assumption is not appropriate, we can use a map graph that generates blank nodes or use unique numeric IDs.

4 Serializing a Map Graph as RDF

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 *map graph* with standard RDF serializations like RDF/XML or N3 because it enables people who are familiar with RDF to edit the *map graph* manually or with some existing popular RDF tools. After we serialize the *map graph* to a file, we can publish the file online to encourage reuse.

There are two subtleties about serializing a *map graph* with RDF123 expressions. First, 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. The second 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. A property with empty local name is not permitted in

RDF/XML serialization, but 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. It is not necessary for a *map graph* to exactly follow RDF syntax because it is just a template rather than a true RDF document. A serialized file of the *map graph* in Figure 3 is shown below.

```
<rdf:RDF
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:foo="http://www.foo.org/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <foo:Club rdf:about="#UMBC CS Friday Afternoon Research Club"/>
  <rdf:Description rdf:about="Ex:$1">
    <rdf:type rdf:resource="Ex:foo+@If($4='Yes';'Professor';'Student')"/>
    <foaf:mbox>Ex:'mailto:'+$2</foaf:mbox>
    <foaf:name>Ex:$1</foaf:name>
    <foo:officeNumber>Ex:$3</foo:officeNumber>
    <foo:hasCoffeeDue>Ex:$5^^decimal</foo:hasCoffeeDue>
    <foo:advisor>
      <foo:Professor rdf:about="Ex:$6"/>
    </foo:advisor>
    <foo:isMemberOf rdf:resource="#UMBC CS Friday Afternoon Research Club"/>
  </rdf:Description>
</rdf:RDF>
```

5 Incorporating Metadata

RDF123 allows people to specify metadata both in map file and spreadsheets. The metadata serves two functions. One is to provide parameters to the translation procedure, such as specifying the spreadsheet region containing the table to be translated, whether the table has a header, and the map file's URI. The other is to add RDF descriptions to the produced RDF graph, such as title, author, and comment. Besides functioning as annotations, the descriptions also provide an identifier via a map file or spreadsheet template to facilitate discovering and collecting a certain type of RDF documents on the Web using a search engine like Swoogle [9] or Sindice [10]. It is possible that metadata specified in the map file can conflict with the one specified in the spreadsheet. When this occurs, if the map file exists as an embedded link in an online spreadsheet, the metadata of the spreadsheet will override the one in the map file because the transformation is controlled by the spreadsheet owner. If a map file is applied to other people's online spreadsheets, the metadata of the map file will override the one in the spreadsheets because the transformation is invoked by the map file owner.

5.1 Metadata in a spreadsheet

RDF123 allow users to specify metadata in a spreadsheet. In this case, users should be owners or co-authors of the spreadsheet. Unlike the case where metadata is specified in a map file, an embedded URL to the online map file is required.

File Edit Table

	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				
19				

Fig. 4. 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.

Spreadsheet metadata is embedded into a contiguous and isolated tabular area with two columns and a header 'rdf123:metadata'. When the RDF123 application or service processes a spreadsheet, it first scans all cells for a recognizable metadata block. If one is found, the RDF123 metadata is extracted, used in the translation process and stored in the resulting RDF graph. If no block is found, the entire spreadsheet is considered as a regular table with the first row being the header row.

In the RDF123 metadata area, people are allowed to tag the spreadsheet in a manner reminiscent to machine tags [11]. The value of a tag can be a literal string or a RDF resource. Some common tags are recognized without defining namespaces using a predefined mapping to 'machine tags'. For example, the 'comment' tag is interpreted as 'rdfs:comment'. For additional convenience, RDF123 also predefines the prefixes of popular namespaces, such as 'rdf', 'rdfs', 'owl', 'dc', 'foaf', 'sioc', 'vcard', and 'swrc'. Figure 4 shows an example of embedded metadata.

5.2 Metadata in the *map graph*

The following is an example for specifying metadata in a map file. The RDF123 expression '*Ex:?*' stands for the base URI of the online RDF document to be translated to. The properties defined in the namespace 'rdf123', such as 'rdf123:startRow' and 'rdf123:endRow' are used to specify the translation metadata. But you can also create annotation metadata or identification metadata by making RDF descriptions about '*Ex:?*'.

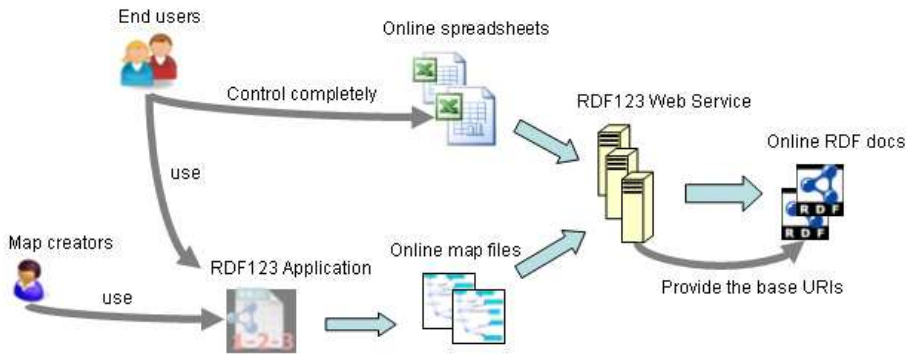

Fig. 5. RDF123 provides an application that allows users to create and edit maps and to generate RDF documents from spreadsheets as well as a Web service that generates RDF documents on demand from online spreadsheets.

```
<rdf:RDF
    xmlns:emp="http://emp.example.org/"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf123="http://rdf123.umbc.edu/ns/">
  <rdf:Description rdf:about="Ex:?">
    <rdf123:startCol>3</rdf123:startCol>
    <rdf123:startRow>6</rdf123:startRow>
    <rdf123:endRow>9</rdf123:endRow>
    <rdfs:comment>use metadata in a map file</rdfs:comment>
  </rdf:Description>
  <foaf:Person>
    <foaf:name>Ex:$1^^string</foaf:name>
    <emp:supervisor>
      <foaf:Person>
        <foaf:name>Ex:$4^^string</foaf:name>
      </foaf:Person>
    </emp:supervisor>
  </foaf:Person>
</rdf:RDF>
```

6 RDF123 Architecture

As shown in Figure 5, RDF123 consists of two components: the RDF123 application and Web service. The application allows users to create and edit RDF123 maps as well as to generate RDF documents from local spreadsheet files. The Web service is designed to automatically generate RDF documents from online spreadsheets in any of several forms using RDF123 maps specified in the service or the spreadsheet itself.

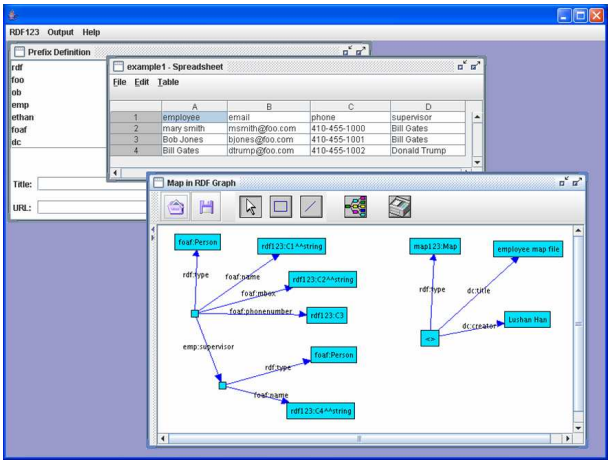

Fig. 6. An RDF123 application (downloadable from [12]) is available for Windows and Unix that provides a graphical interface for creating, inspecting and editing RDF123 maps and using them to generate RDF documents from local spreadsheets.

6.1 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 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 a *map graph*.

The RDF123 application shown in Figure 6 is composed of three internal frames. The first, 'prefix definition', works as a prefix library in which users store (namespace, prefix) pairs. Namespaces are typically long and forgettable URLs that are hard to manage for many users. By using the 'prefix definition' list, users need not write the full namespace again but use prefix wherever a namespace is required. The second frame is a spreadsheet editor, which enable users to open a CSV file, edit, and save the file. The third frame is an interactive graphical editor that allows user to create and remove a vertex/edge, drag a vertex, and change properties of a vertex/edge. With this graph editor, users create their *map graph*s and saved them to local files, along with the positions of the vertices/edges, for the purpose of future modification. They can also be serialized as RDF documents in RDF/XML or N3.

6.2 RDF123 Web Service

The RDF123 Web service is a public service that translates online spreadsheets to RDF and also works as the host of the URIs of the produced RDF documents.

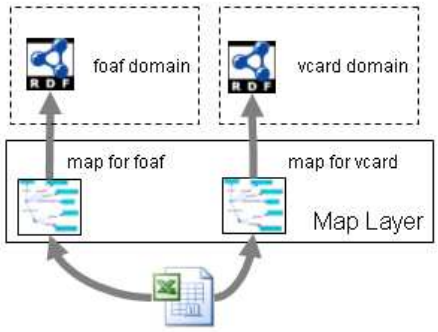

Fig. 7. Separating the spreadsheet data and the maps used to convert them to RDF makes it easy to generate different RDF encodings from the same spreadsheet, encouraging data reuse.

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/`. Note the the RDF123 Web service need not be a centralized service and should, in fact, be replicated by different individuals and organizations.

6.3 RDF123 Map Layer

Adding a map layer between the original data in spreadsheets and converted data in RDF can smooth data reusability and maintenance. People may have different aspects and interests in interpreting spreadsheet data. By using different RDF123 maps, the same data can be available in different domains just by associating it with different map files. Figure 7 gives an example. Moreover, when the domain ontology evolves, the map file can be modified, rather than the physical RDF documents, in order to have the data adapt to the change Thus, data maintenance is eased, since data is directly maintained by spreadsheet owners and the RDF data is always rendered current.

In other cases, the map layer can also play a role in integrating 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 usually have different formats and duplicated data. In order to conduct statistical analyses, researchers must do considerable pre-processing to ensure the data have the same format. With the

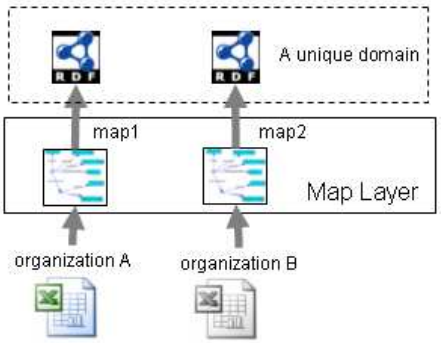

Fig. 8. Separating the data and maps also enables different organizations to have spreadsheet data in their own unique formats but mapped to the same RDF ontology.

help of RDF123, researchers 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, they can load converted RDF data to a triple store and use a SPARQL query to output data in tabular format which can be accepted by a statistical analysis program. Figure 8 gives an illustration.

6.4 A Case Study

We evaluated an early version of RDF123 during the first annual Blogger Bioblitz [13, 14]. A Bioblitz [15] is a 24 hour long inventory of the living organisms in a given location, typically by a team of scientists leading a larger group of students and hobbyists with the goals of raising interest in and awareness about biological diversity.

We found that this application demonstrated the strength of RDF123 as a means of publishing and collating distributed data maintained in public spreadsheets. Participants were bloggers who spent one day observing as many different organisms in their chosen location(s). Last year, a common spreadsheet of observations was completed by each individual blogger and then sent to a central location for collation. Idiosyncrasies in the way the spreadsheets were completed made manual integration into a single spreadsheet difficult, but we did successfully use RDF123 to make over 1500 observations available in RDF. However, the data could not easily be corrected or updated. In the 2008 Blogger Blitz we will urge participants to maintain their data publicly using Google spreadsheets. Some map files may need to be created to capture idiosyncrasy, but this should be easier and more dynamic than reformatting to a common template.

6.5 Publishing and Harvesting RDF Data from Spreadsheets

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 [9].

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 and has indexed over 1,350,000 CSV files and 14,700,000 XLS files. If we use a search engine to query for spreadsheet files using keywords that are particular to RDF123 metadata like 'rdf123:metadata', 'map file' and tag values, 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 can find them via Google API and convert them to RDF without the involvement of end users.

7 Conclusions and Future Work

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 online.

Our experience in using RDF123 in the 2007 Blogger Bioblitz convinced us that RDF123 did a good job in meeting our design goals. Users who were familiar with spreadsheets and general computer applications found the application and its tools for modeling data both easy to understand and use. Bioblitz participants found their familiar spreadsheet systems convenient for entering and editing data. The flexibility of the RDF123 mapping language allowed more sophisticated users (i.e., the authors) to refine and publish the maps to produce the desired target encoding in RDF.

There is still one barrier left for common users to use RDF123 to contribute their data to the Semantic Web. Although drawing a *map graph* in the RDF123 application is not hard, choosing proper Semantic Web terms (classes and properties) requires familiarity with appropriate ontologies and the terms they define. We are developing a system that suggests appropriate RDF terms given semantically related English words and general domain and context information [16]. The Swoogle Semantic Web search engine is used to provide RDF term and namespace statistics, the WordNet lexical ontology to find semantically related words, and a naive Bayes classifier to suggest terms. Our initial results show good performance in predicting appropriate RDF terms as measured by precision and recall and we are optimistic that it will be a useful extension to the RDF123 application.

References

1. Hazael-Massieux, D., Connolly, D.: Gleaning Resource Descriptions from Dialects of Languages (GRDDL), W3C Team Submission. Technical report, World Wide Web Consortium (May 2005)
2. Reck, R.P.: Excel2rdf for microsoft windows. `http://www.mindswap.org/\~rreck/excel2rdf.shtml` (January 2003)
3. Grove, M.: Mindswap Convert To RDF Tool. `http://www.mindswap.org/\~mhgrove/convert/`
4. Golbeck, J., Grove, M., Parsia, B., Kalyanpur, A., Hendler, J.: New Tools for the Semantic Web. Proceedings of the European Knowledge Acquisition Workshop (2002) 392–400
5. Huynh, D., Karger, D., Miller, R.: Exhibit: lightweight structured data publishing. In: Proceedings of the 16th international conference on World Wide Web, ACM Press New York, NY, USA (2007) 737–746
6. TopQuadrant: TopBraid Composer Web site. `http://www.topbraidcomposer.com/`
7. Bizer, C., Seaborne, A.: D2RQ-Treating Non-RDF Databases as Virtual RDF Graphs. Proceedings of the 3rd International Semantic Web Conference (ISWC2004) (2004)
8. Bizer, C.: D2R MAP-A Database to RDF Mapping Language. Proceedings of the 12th International World Wide Web Conference (May 2003) 20–24
9. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V.C., , Sachs, J.: Swoogle: A search and metadata engine for the semantic web. In: Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management. (2004)
10. Tummarello, G., Delbru, R., Oren, E.: Sindice. com: Weaving the open linked data. In: Proceedings of the Sixth International Semantic Web Conference, Springer (November 2007)
11. Schmitz, P.: Inducing ontology from Flickr tags. In: Proceedings of the WWW2006 Collaborative Web Tagging Workshop. (May 2006)
12. : RDF123 Web site. `http://rdf123.umbc.edu/`
13. Scienceblogs: Blogger bioblitz. `http://scienceblogs.com/voltagegate/2007/03/announcing_the_first_annual_bl.php`
14. Parr, C., Sachs, J., Han, L., Wang, T., Finin, T.: RDF123 and Spotter: Tools for generating OWL and RDF for biodiversity data in spreadsheets and unstructured text. In: Proceedings of the Biodiversity Information Standards (TDWG) Annual Conference 2008. (October 2007) (poster abstract).
15. Lundmark, C.: BioBlitz: Getting into Backyard Biodiversity. BioScience **53**(4) (2003) 329–329
16. Han, L., Finin, T.: Predicting Semantic Web Terms from Words. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI Press (July 2008) (student abstract).