XLWrap – Querying and Integrating Arbitrary Spreadsheets with SPARQL

Andreas Langegger and Wolfram Wöß

Institute of Applied Knowledge Processing
Johannes Kepler University Linz
Altenberger Straße 69, 4040 Linz, Austria
{al,wolfram.woess}@jku.at

Abstract. In this paper a novel approach is presented for generating RDF graphs of arbitrary complexity from various spreadsheet layouts. Currently, none of the available spreadsheet-to-RDF wrappers supports cross tables and tables where data is not aligned in rows. Similar to RDF123, XLWrap is based on template graphs where fragments of triples can be mapped to specific cells of a spreadsheet. Additionally, it features a full expression algebra based on the syntax of OpenOffice Calc and various shift operations, which can be used to repeat similar mappings in order to wrap cross tables including multiple sheets and spreadsheet files. The set of available expression functions includes most of the native functions of OpenOffice Calc and can be easily extended by users of XLWrap.

Additionally, XLWrap is able to execute SPARQL queries, and since it is possible to define multiple virtual class extents in a mapping specification, it can be used to integrate information from multiple spreadsheets. XLWrap supports a special identity concept which allows to link anonymous resources (blank nodes) – which may originate from different spreadsheets – in the target graph.

1 Introduction

The translation of information stored in various legacy information systems and data formats to RDF is an important requirement of many Semantic Web applications. While for the Web of Data relational database management systems (RDBMS) are considered to be the most important legacy information systems, in case of corporate Semantic Web applications, spreadsheets play a similar important role. Spreadsheets are frequently used by people in companies, organizations, and research institutions to share, exchange, and store data. Whenever there is no database in place, spreadsheets are often the primary fall-back tool for maintaining structured information.

Compared to wrapping a relational database, which has a fixed schema, data types, and integrity constraints, wrapping spreadsheets is more difficult because the implicit schema has to be captured first when creating a formal mapping. Currently available spreadsheet wrappers treat spreadsheets as flat tables like single database relations or comma-separated value (CSV) files. In this paper we will present a novel mapping approach for spreadsheets which is based on template graphs similar to RDF123 [5], which we will explain in the related work part. However, the XLWrap mapping approach is not based on a simple row oriented iteration of tables. It allows to define template mappings as RDF graphs and to repeat them based on various shift and repeat operations in order to map arbitrary layouts including multi-dimensional cross tables and spreadsheets over multiple files. XLWrap supports expressions to reference cells and ranges from template graphs including sheet ranges and absolute references to other external spreadsheets such as supported by Microsoft Excel and OpenOffice Calc (the grammar is printed in Listing 1 on page 368).

Additionally, XLWrap features a server component called *XLWrap-Server*, which provides a SPARQL endpoint as well as a linked data browser similar to D2R-Server [3]. XLWrap-Server observes a configurable directory for mapping files and whenever a mapping file or one of the referred spreadsheet files of the mapping is added, modified, or removed, it automatically executes the translation process and caches the generated RDF data. XLWrap-Server integrates *Joseki* [6] to provide a SPARQL endpoint and *Snorql*, which has been adopted from D2R-Server and allows the user to explore the wrapped information on-the-fly. Together with the *Semantic Web Integrator and Query Engine* (SemWIQ) [7], XLWrap can be used to integrate various spreadsheets and other data sources such as relational databases and ODBC data sources. A simple setup procedure and ease of use have been two major requirements for the development of XLWrap-Server. The setup procedure is a matter of starting the server and putting mapping files into the observation folder. XLWrap-Server is also considered to become a practical tool for experimenting with linked data. It can be used to quickly expose information via SPARQL while editing it in a human-friendly way. XLWrap can be downloaded from its homepage at `http://www.langegger.at/xlwrap`

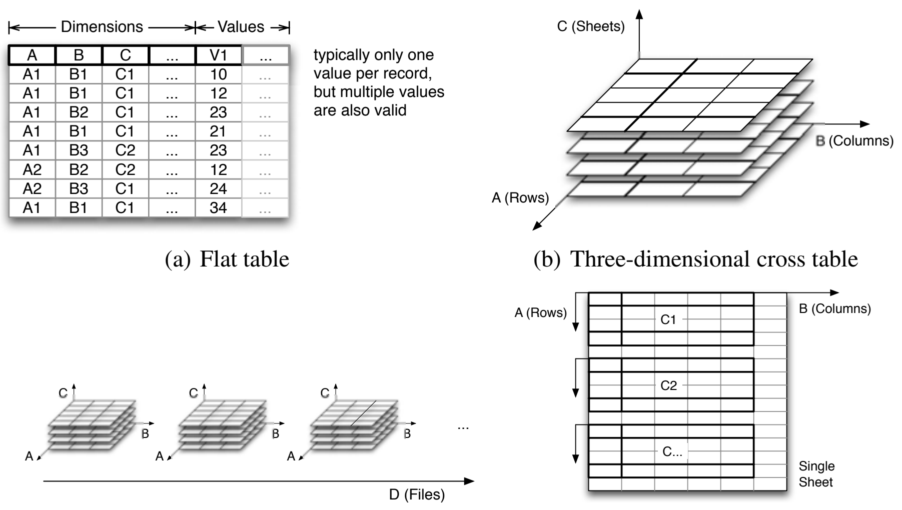
In the next section we will explore different spreadsheet layouts in order to create a sound mapping framework. In Section 3 related work is discussed. In Section 4 the XLWrap mapping formalism is presented and in Section 5 the transformation process is explained. In order to demonstrate XLWrap, an example is discussed as part of these sections. Section 6 concludes the contribution with some final remarks.

2 Background

In order to develop a generic mapping framework for spreadsheets, the spreadsheet paradigm has been analyzed and different ways of information representation have been examined. Some of these findings are presented as part of this section before related work will be discussed. In the following, we will use the terms *workbook* to refer to a file of a spreadsheet application, and *worksheet* (or just *sheet*) to denote a single two-dimensional sheet of a workbook.

2.1 Information Representation in Spreadsheets

Firstly, it is important to distinguish between the *information model* and the *representation model*, which is used to represent information within a spreadsheet. The *information model* is defined implicitly by the semantics of the entailed information. The resulting RDF graph should as closely as possible reflect the information model, as for example, expenditures by category, year, and sub-division or personal information about employees. The actual information representation as an RDF graph is a subject of



A	B	C	...	V1	...
A1	B1	C1	...	10	...
A1	B1	C1	...	12	...
A1	B2	C1	...	23	...
A1	B1	C1	...	21	...
A1	B3	C2	...	23	...
A2	B2	C2	...	12	...
A2	B3	C1	...	24	...
A1	B1	C1	...	34	...

(a) Flat table

(b) Three-dimensional cross table

(c) Four-dimensional cross table

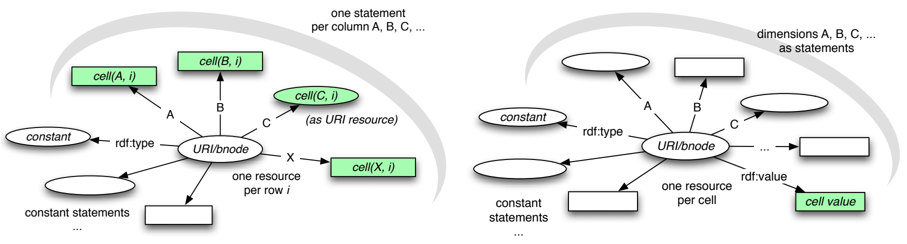
(d) Cross table on single sheet

Fig. 1. Information representation in spreadsheets

data modeling and in fact, there are many ways how to represent expenditures in RDF. A generic wrapper should not enforce any fixed rules and structures that depend on the representation of the information model in the spreadsheet. Concerning the *representation model*, three different layouts could be identified, whereas the third one is a hybrid approach of the first two:

One-Dimensional Flat Table. In this layout (Figure 1(a)) information is represented, regardless of its dimensionality, in a flat table with a single column (*or* row) heading. It is used to represent information such as data lists, e.g. persons with fixed properties such as name, mailbox, age, etc. Except for RDF123, all existing wrappers create exactly one RDF resource per row as shown in Figure 2(a). Currently available spreadsheet wrappers, which will be discussed in Section 3, are restricted to this kind of representation.
However, flat tables can also be used to represent information with multiple dimensions (typically >2) in so-called de-normalized tables which are also the basis of pivot tables and OLAP programs. Some cells of the header represent dimensions of the information model, the other header cells represent values specific to each instance. The domain values of dimensions repeatedly occur in the instance rows as shown in Figure 1(a). Note that regarding the previous example of a person data list, no dimensions will be used and each value is regarded as specific to the person instance. Alternatively, each person property may be regarded as a dimension or *facet* (for instance, the dimension *first name* may have all occurring first names as its domain).

Cross Tables. In this layout, which is shown in Figure 1(b), the representation of information is organized in cross tables, which may span multiple columns, rows,



(a) One RDF resource per row (b) One RDF resource per cell

Fig. 2. Straight-forward translation of spreadsheet data to RDF resources

sheets, and even files and directories (Figure 1(c)). Each cell represents a single entity or instance. In a straight-forward approach, the translation could be done as shown in Figure 2(b). Instead of a single column/row header, cross tables have multiple headers, one for each dimension. Along the sheet axis, the header is defined either by the names of the sheets in the workbook or by text labels placed into the worksheet. Similarly, when using multiple files, the domain values of the corresponding axis are either defined by file names or text labels on worksheets. Because a single sheet is already restricted to columns and rows, cross tables are often represented by repeating similar table layouts on the same sheet as depicted in Figure 1(d). Similar to pivot tables or de-normalized tables as mentioned before, a higher dimensionality is broken down into two dimensions by repeating similar structures.

Hybrid Layouts. Finally, there is the possibility of combining flat tables with cross tables (e.g. de-normalized tables can be repeated across several sheets or files).

Independent of the representation model, a wrapper must be able to translate the information into an RDF graph by best reflecting the source information model. To give another example, revenues for different products, years, and countries can either be represented in a de-normalized flat table with the heading *(product, year, country, revenue)* or in a cross table with one sheet per *product*, each one representing *revenues* per *year* and *country*.

2.2 Definition of Spreadsheet Applications

The following definition for spreadsheet applications is the basis for the mapping framework presented in Section 4.

Definition 1. *A spreadsheet application A is defined as a set of named workbooks w_i, $i \in \mathbb{N}_0$, where* name(w_i) *is the canonical workbook filename denoted by its URI. A spreadsheet application has at least one workbook w_0, called the* base workbook *and optionally a set of* external workbooks w_i, $i \geq 1$, *which are referenced from w_0 by means of absolute cell references.*

Within companies it is very common that multiple spreadsheets are interlinked by external cell references. To be able to process external references in XLWrap mappings, this definition takes external workbooks into account. The URI scheme for local files is `file://`. Workbook files on the Web can be referenced by means of HTTP URIs.

Definition 2. *A workbook $w_i \in A$ is defined as a set of named worksheets s_j, $j \in \mathbb{N}$, where* name(s_j) *is the sheet name as specified in the application (a string).*

Beside single worksheets, it is also common to use multiple worksheets. As will be shown later, XLWrap supports a special sheet shift operation, which allows to repeatedly apply template mappings on multiple sheets.

Definition 3. *A worksheet $s_j \in w_i$ is defined as a matrix of cell values $V = [v_{c,r}]_{n \times m}$ where $c \in \mathbb{N}_0$, $c < m$, is the column index and $r \in \mathbb{N}_0$, $r < n$, is the row index of the corresponding worksheet table. m, n denote the total number of columns and rows used in worksheet s_j.*

Although there is a space limit for m and n in practice, there is no limitation enforced by this definition. The same applies to the number of worksheets in a workbook.

Definition 4. *A cell value $v_{c,r}$ has a* type annotation *denoted as* type$(v_{c,r}) \rightarrow T$ *with* $T = \{t_{text}, t_{number}, t_{datetime}, t_{boolean}, t_{empty}\}$. *Additionally, a cell value may have a* formatting *(e.g. for number, currency, date/time formatting) and a* formula annotation *denoted as* formula$(v_{c,r}) \rightarrow E$, *where $e \in E$ is a compositional expression according to a predefined grammar G_E.*

A cell formula, is not a specific type $\in T$. Instead, it is an annotation defining the expression which can be evaluated to reproduce or update $v_{c,r}$. Values with formula annotations are explicitly stored in the corresponding workbook files. In our definition of spreadsheet applications, the grammar G_E supports *range references* denoted as e_{ref}. Range references are used to refer to other cells of the same spreadsheet, other sheets of the same workbook, or external workbooks. In Section 4 it will be described, how expressions and range references are also used for XLWrap mappings to map spreadsheet cells to RDF graphs. The proper definition of a *range reference* is given below.

Definition 5. *A range reference sub-expression e_{ref} is generally defined as a set of partial 7-tuples of the form $(w_i, s_{j1}, c_1, r_1, s_{j2}, c_2, r_2)$. These components specify a workbook, a start cell consisting of a sheet, column, row, as well as a target cell specified by a sheet, column, and row. Some tuple components are optional and may be left empty. If $|e_{ref}| > 1$, the range reference is called* multi range reference, *because it contains multiple single range references. In case of OpenOffice Calc, the general lexical representation of a single range reference is:*

$$(\ (\ w_i\ \text{“\#\$”}\ )?\ s_{j1}\ \text{“.”}\ )?\ c_1\ r_1\ (\ \text{“:”}\ (\ s_{j2}\ \text{“.”}\ )?\ c_2\ r_2\ )?$$

For multi range references, single range references are separated with a semicolon. The following two sub-types are defined:

- Cell reference*, which has the optional components w_i, s_{j1}, the mandatory components c_1, r_1, and the component s_{j2}, c_2, r_2 left empty (e.g.* `A3`*,* `'Sheet 1'.A3`*, or* `file:foo.xls#$'Sheet 1'.A3`*)*
- Box reference*, which has the optional components w_i, s_{j1}, s_{j2} and the mandatory components c_1, r_1, c_2, r_2 (e.g.* `A3:B9`*, or* `file:foo.xls#$'Sheet 1'.A3:B28`*.*

Whenever the components w_i and s_{j1}, s_{j2} are omitted, the range reference is interpreted relative to the worksheet of its originating cell formula. Hence, a range reference can either be *absolute* or *relative* its base sheet or workbook file.

2.3 Dumping versus On-the-Fly Processing of SPARQL Queries

Concerning the wrapping approach, a distinction can be made based on the direction of the mapping formalism, which can either be source/spreadsheet-centric, or target/RDF-centric. In general, when mapping between two different data models, it is possible to define one of the data models as a view onto the other data model. Since in our case, the RDF model acts as the target model, the spreadsheet-centric variant is similar to the *Local-as-View* (LaV) approach and the RDF-centric variant is similar to the *Global-as-View* approach (GaV) in information integration [8]. Only XLWrap and RDF123, support the GaV-approach and allow the user to define RDF-centric mappings based on graphs. All other wrappers are based on a spreadsheet-centric view definition which map columns or cells to specific properties and do not allow the definition of custom target graphs.

Another distinction can be made concerning the actual transformation of spreadsheet data into RDF when executing SPARQL queries. For queries, the wrapping process can either be based on materialization involving a single data dump into a volatile or persistent RDF store, or it can be an on-the-fly query execution process (e.g. D2R-Server [3]). Actually, our initial motivation for developing XLWrap have been experiments towards a generic guideline for the generation of virtual RDF wrappers supporting SPARQL for the *Semantic Web Integrator and Query Engine* (SemWIQ). Based on experiences with D2R-Server while contributing some optimizations like push-down of filters into SQL, our assumption was, that simple RDF wrappers can be written based on Jena ARQ by supporting triple patterns and let ARQ process all higher level algebra operations. For instance, a wrapper providing SPARQL access to the directory structure of a file system or an FTP server can be written very easily this way. It only needs to correctly interpret subsequent triple patterns and generate the necessary variable bindings which correspond to a virtual RDF graph similar to a database access plan. However, for larger information sources, low-level index structures such as hash tables or B+ trees are necessary in order to achieve acceptable performance results. They may be managed by the wrapper in volatile memory caches and will have to be updated by means of a notification mechanism or pre-defined periods. Additionally, a wrapper for a large data source should provide cardinality estimations for triple patterns in order to support the re-ordering in a BGP by expected cardinalities. During the evaluation of a BGP, subsequent joins over triple patterns are executed by ARQ based on a substitute algorithm: already obtained variable bindings of triples produced by upper triple patterns are used as constraints for matching subsequent triple pattern. Thus, triple patterns with a lower cardinality should be executed first.

However, experiments have shown that in most situations it is still better to apply a materialized approach and dump data into an RDF store instead of maintaining separate index structures and insisting on the virtual approach. In particular, these findings apply to spreadsheets, because they are typically small in size (compared to databases) and there is actually no need to process queries on-the-fly. In fact, the dumping process is so fast, that even if the materialized cache has to be re-generated, it is only a matter of seconds or milliseconds. XLWrap tracks any changes in mapping files and referenced workbook files and re-generates caches in the background which is hardly noticed by end-users.

3 Related Work

Among the currently available spreadsheet wrappers there are two open source projects, *Excel2RDF* [4] and *RDF123* [5], and there is *TopBraid Composer* from TopQuadrant[1], which is a commercial ontology development environment integrating support for importing and exporting RDF from/to spreadsheet files. We were unable to find any further spreadsheet RDF wrapper in the literature and on the Web. There is another notable software called *Anzo for Excel* from Cambridge Semantics[2], which is actually not a dedicated RDF wrapper, but which can be used to map information from spreadsheets to vocabularies for distributed, collaborative work. Because Anzo is targeted to employees who typically do not know about RDF, it hides any details of the internals. Although it provides a graphical plugin for Microsoft Excel, which allows to map cells and ranges to vocabularies, it is unclear how it can be used to map spreadsheets to custom graphs. As a marginally related project the *Aperture*[3] framework developed for the *Nepomuk Semantic Desktop* is mentioned, which is extracting meta data from spreadsheet files. However, the content of spreadsheets is not processed or wrapped based on a formalized mapping. Instead, Aperture tries to extract significant features and terms from spreadsheets in order to support semantic search and interlinking of documents in the Nepomuk framework.

Unfortunately, nearly all existing dedicated wrappers are rather limited in practice. The simplest one is Excel2RDF and an extended version called *ConvertToRDF*, which supports basic mappings for column headers. As both tools follow the spreadsheet-centric mapping approach, the output graph is fixed and for each row of only a single spreadsheet table, one RDF resource is created with property/object pairs for all mapped columns. Resources as objects or typed literals are not supported. The spreadsheet import feature of TopBraid Composer is similarly basic. Actually, only CSV text files are supported and the mapping is similar to ConvertToRDF. Typed literals and resource objects are supported, but target types have to be specified manually.

The most relevant wrapper related to our work is RDF123 [5]. Beside XLWrap, it is the only wrapper that supports an RDF-centric mapping approach. However, although with RDF123 it is possible to define arbitrary target graphs, it is restricted to a specific spreadsheet layout, like all of the other existing wrappers. It only supports flat tables

[1] `http://www.topquadrant.com`
[2] `http://www.cambridgesemantics.com/`
[3] `http://aperture.sourceforge.net/`

as shown in Figure 1(a) of the previous section. The available prototype is capable of reading CSV text files but does not support Excel or OpenOffice spreadsheets. An RDF mapping in RDF123 is described as part of the spreadsheet itself in a special meta data section starting with the specific label `rdf123:metadata`. Among several Dublin Core metadata, the section specifies the start row, the end row, the start column, whether there is a row header which has to be skipped, and the URI of the template graph used for producing the results. When processing a spreadsheet, the execution engine scans the sheet for the metadata section and if it cannot be found, the complete sheet (only one worksheet is supported) is wrapped in a similar way as with Excel2RDF. If the metadata section is found, the execution engine retrieves the template graph from the specified location. The template graph is an RDF graph which may contain references to the columns of the current row being processed to generate nodes based on values obtained from the spreadsheet. Thus, the execution process is fixed to one row by row iteration. In order to refer to the columns of the active row being processed, expressions can be specified as special literals and faked URIs as for example `"Ex:$1"` and `<Ex:$1>`, which both refer to the first column. Unfortunately, this approach has the following severe consequences: all other literals starting with `Ex:` will be matched as expressions and cause errors. Furthermore, because expressions may contain spaces and special characters, encoding them as URIs (with `Ex:` as a pseudo protocol) will cause further troubles at runtime and lead to incompatibilities with other systems.

RDF123 provides a GUI which facilitates the process of creating mappings. However, the tool only provides a viewer for CSV files and a basic graph designer which both are not combined in any special way. The graph designer uses a proprietary format to store graphs specific to the Open JGraph library, which has been used. Developing a powerful graphical support tool is not easy and requires a lot of effort. It would be desirable for a graphical mapping tool to provide features specific to the mapping process such as drag and drop, highlighting of referenced cells, and a simulation feature for debugging the wrapping process.

Because XLWrap can be used to semantically integrate different spreadsheets and to query the entailed information based on logical inference, it can also be compared to the system proposed by [9] which is called LESS for *Logic Embedded in SpreadSheets*. Instead of writing typical numerical formulas into the cells of a spreadsheet, LESS allows to logically annotate facts and use logical functions for calculations.

4 XLWrap Mapping Formalism

XLWrap is based on an RDF-centric mapping approach which allows to map the information stored in a spreadsheet to arbitrary RDF graphs independent from the representation model as discussed in Section 2. Similar to RDF123, the output graph is defined by means of template graphs which are repeatedly applied during the wrapping process. With XLWrap expressions it is possible to refer to arbitrary cells and combine them with expressions like in spreadsheet applications.

4.1 XLWrap Mappings

In the following, XLWrap mappings are defined based on the definitions of Sect. 2.

Definition 6. *An XLWrap mapping M is defined as a set of* map templates $m_k, k \in \mathbb{N}$.

Definition 7. *A* map template $m_k = (w_k, s_k, C_k, G_k, F_k)$ *consists of the following components: a base workbook $w_k \in w_i$, a base worksheet $s_k \in s_j$, a constant RDF graph C_k, an RDF* template graph G_k, *and a sequence of* transform operations $F_k = (f_l)$, $l \in \mathbb{N}$.

Definition 8. *A* constant graph C_k *and a* template graph G_k *are valid RDF graphs, according to the W3C specification, which may contain literals of the custom data type* `xl:Expr` *called* XLWrap expressions[4].

Definition 9. *A transform operation f_l can modify the template graph G_k and change range references in expressions.*

During the wrapping process, each map template $m_k \in M$ contributes a sub-graph $[[G_k]]$ to the overall result similar to RDF123, but with the difference that a graph template is not moved from the first row to the last one in a fixed direction, instead, it is moved based on the transformation sequence defined by (f_l). The bracket notation $[[\ldots]]$ is used to denote the application of a template graph including the evaluation of all XLWrap expressions. While C_k is evaluated and merged into the target graph once, the template graph G_k is subsequently transformed by (f_l) and evaluated multiple times. If $|F_k| = 0$, no transformations are specified and G_k is applied once in its initial form.

Because by definition subjects and predicates cannot be literals, in order to specify XLWrap expressions for subjects or predicates, they have to be wrapped in blank nodes as part of the template graph. The special property `xl:uri` is then used to replace these blank nodes by URI resources in the target graph. Similarly, the property `xl:id` can be used to link blank nodes: XLWrap will produce blank nodes with equal local IDs in the target graph

Definition 10. *An XLWrap expression is defined as a compositional expression of basic expression symbols similar to spreadsheet formulas (see Definition 4). XLWrap expressions are parsed from the lexical representation by the custom data type implementation*[5] *according to the grammar defined in Listing 1.*

Range references, including cell, box, and multi range references are supported as specified in Definition 5. Additionally, XLWrap supports the special range types *null range*, *full sheet range*, and *any range*, which are lexically denoted as the empty string, (s_j ".*"), and "*.*". Depending on the semantics of operations and functions, only specific range sub-types are valid[6]. Optionally, a worksheet s_j can be prefixed with "#" and specified by the sheet number starting with 1 (e.g. "#1.A1", "#3.*").

XLWrap supports all standard arithmetic and also logical operators, string concatenation, and an extensible function library (additional implementations can be easily added at runtime). The most important functions for string manipulation, including SHA-1

[4] The full namespace for the prefix `xl:` is `http://langegger.at/xlwrap/vocab#`

[5] In Jena it is possible to register a custom data type handler by extending `BaseDatatype`.

[6] For example, while `SUM()` takes any range reference and also numbers as arguments (the number of arguments is not restricted), the expression `"A3:B5"` is invalid, since it is only possible to obtain values from a single cell.

```
XLExpression   =  "="? OrExpr <EOF> OrExpr        =  AndExpr (
"||" AndExpr )* AndExpr        =  Comparable ( "&&" Comparable )*
Comparable     =  Concatable ( CompOp Concatable )* Concatable =
Expr ( "&" Expr )* Expr          =  Term ( ("+"|"-") Term)* Term
=  Factor ( ("*"|"/") Factor)* Factor        = Atom ("^" Atom)*
Atom           =
   ("+"|"-"|"!")
   (
      <NUMBER> ("%")? |
      (<TRUE>|<FALSE>) |
      <STRING> |
      <CELLRANGE> |
      "(" Concatable ")" ("%")? |
      <FUNCIDENT> "(" ( Concatable ( (","|";") Concatable)* )? ")" ("%")?
   )
CompOp         =  "<=" | "<" | ">=" | ">" | ("!="|"<>") | ("=="|"=")
```

Listing 1. Grammar of XLWrap expressions

hashing (which, for instance, is required for `foaf:mbox_sha1sum` property values in FOAF applications), type casting (to enforce specific literal data types in the RDF output), aggregate functions such as `SUM()`, which takes cell, box, and multi ranges as arguments, have already been implemented.

The following transform operations are available in the current implementation:

- column shift: $f_{ColumnShift}(d, n, z, e_c)$
- row shift: $f_{RowShift}(d, n, z, e_c)$
- sheet shift: $f_{SheetShift}(d, n, z, e_c)$
- sheet repeat: $f_{SheetRepeat}((g_i), z, e_c)$
- file repeat: $f_{FileRepeat}((h_i), z, e_c)$

Common to all operations is z, a multi range reference, which can be used to restrict the transform operation on a set of ranges (default is `AnyRange`) and e_c can be a logical XLWrap expression, which is evaluated each time before the transformation is applied (default is *true*). For all the *shift* operations d is the amount of columns/rows/sheets to shift (defaults to 1), n is the number of times to apply the operation (defaults to the maximum integer value of the runtime system), and for the repeat operations, (g_i) and (h_i), respectively, specify the set of sheets or files to apply the template for. In order to dynamically wrap evolving spreadsheets, n can be omitted and the iteration can be controlled based on the condition e_c. As a consequence, the transform operation will be repeated until the condition evaluates to false. For convenience, the special function `EMPTY(`e_{ref}`)`, which takes a multi range argument and returns true if all cells in e_{ref} are empty, can be used to detect the end of a data range in the spreadsheet.

As mentioned along with Definition 5, a range reference can be *absolute* or *relative*. Relative range references are extended during the mapping process by the base workbook w_k and base worksheet s_k defined in the mapping M. For example, "A3" may refer to "file:foo.xls#$Sheet1.A3" at runtime. The sheet/file repeat transformations will override the sheet/file component as needed, but absolute range references are never modified by transform operations. There are special expression functions which can be used to track the origin of the generated RDF triple by obtaining the current filename, sheet name, sheet number, row and column of a cell at runtime: `FILENAME()`, `SHEETNAME()`, `SHEETNUM()`, `COLUMN()`, and `ROW()`. All these functions take a cell range as an argument.

4.2 Example Mapping

The source data for the example is printed in Table 1. The workbook w_0 used for this demonstration contains two sheets s_1, s_2 of information on revenues of a company. For each country the company operates in, revenues are organized in a cross table containing sold items and total revenue per product and year. As can be seen, data for 2008 is missing for Germany, and there is one more product for Germany.

Table 1. Source data for the discussed example ($s_1, s_2 \in w_0$)

Austria							
	2007		2008		2009		
product	items sold	revenue	items sold	revenue	items sold	revenue	
Product 1	342	7,866.00	376	8,648.00	490	11,760.00	
Product 2	4,333	1,005,256.00	5,655	1,328,925.00	3,493	838,320.00	
Product 3	3,312	1,136,016.00	4,566	1,598,100.00	5,993	1,917,760.00	
Product 4	45	19,350.00	56	24,304.00	54	23,328.00	
Totals	8,032	2,168,488.00	10,653	2,959,977.00	10,030	2,791,168.00	

Germany					
	2007		2009		
product	items sold	revenue	items sold	revenue	
Product1	2,431	55,913.00	3,419	82,056.00	
Product2	31,230	7,339,050.00	32,123	7,709,520.00	
Product3	23,121	8,092,350.00	31,039	9,932,480.00	
Product4	3,423	1,198,050.00	3,412	1,091,840.00	
Product5	121	52,514.00	312	134,784.00	
Totals	60,326	16,737,877.00	70,305	18,950,680.00	

Depending on the desired target graph, $C_k, G_k, andF_k$ can be specified differently. For instance, the target graph could be modeled as one resource per country having linked resources for all products, and years. A more direct representation of the multi-dimensional information is defined in the example mapping shown in Listing 2. We will describe the generation process for this mapping in the next section. A third representation using the *Statistical Core Vocabulary*[7] (SCOVO) is provided as part of the distribution (`mappings/iswc09-example.trig`). In order to be able to include the template graphs in the mapping specification, the TriG syntax[8], which allows to denote named graphs, is used. XLWrap searches for an instance of `xl:Mapping` in all graphs and starts parsing the specification. The XLWrap mapping vocabulary, which is published at `http://www.langegger.at/xlwrap/vocab#`, corresponds to the definitions provided in Section 4.1.

[7] `http://purl.org/NET/scovo`

[8] `http://www4.wiwiss.fu-berlin.de/bizer/TriG/`

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xl:   <http://langegger.at/xlwrap/vocab#> .
@prefix ex:   <http://example.org/> .
@prefix :     <http://myApplication/mapping#> .

{ [] a xl:Mapping ;
    xl:template [
        xl:fileName "files/testing/iswc09-example.xls" ;
        xl:sheetNumber "0" ;
        xl:templateGraph :Revenues ;
        xl:transform [
            a rdf:Seq ;
            rdf:_1 [ a xl:RowShift ;
                xl:restriction "A4; B4:C4" ;
                xl:condition "LEFT(A4, 7) == 'Product'" ;
                xl:steps "1" ] ;
            rdf:_2 [ a xl:ColShift ;
                xl:restriction "B2; B4:C4"^^xl:Expr ;
                xl:condition "!EMPTY(B4:C4)" ;
                xl:steps "2" ] ;
            rdf:_3 [ a xl:SheetShift ;
                xl:restriction "#1.*"^^xl:Expr ;
                xl:repeat "2" ] ;
        ]
    ] .
}

:Revenues {
    [ xl:uri "'http://example.org/revenue_' & URLENCODE(SHEETNAME(A1) & '_' & B2 &
            '_' & A4)"^^xl:Expr ] a ex:Revenue ;
    ex:country      "DBP_COUNTRY(SHEETNAME(A1))"^^xl:Expr ;
    ex:year         "DBP_YEAR(B2)"^^xl:Expr ;
    ex:product      "A4"^^xl:Expr ;
    ex:itemsSold    "B4"^^xl:Expr ;
    ex:revenue      "C4"^^xl:Expr .
}
```

Listing 2. Example mapping specified in TriG syntax

In our approach, the mapping is stored separately from the actual spreadsheet files and not as part of them. It is assumed that the creator of the mapping file may not have the authorization or possibility to modify spreadsheet files to be wrapped. XLWrap is capable of loading local files and downloading spreadsheet files from the Web. Currently, Excel files, Open Document spreadsheets, and also CSV files are supported (although they could also be wrapped with other tools). The layout of CSV files (delimiters, separators, white spaces) can be specified. CSV files are streamed in order to support large data files. The implementation simulates a workbook and requests new rows from the input stream as needed by the execution engine. Because a single template graph typically refers to a small section of the whole spreadsheet, it is sufficient to keep the last n (where $n = 1000$ by default) rows in a cache.

5 Transformation Process

To give an overview of the transformation process, a flow chart is depicted in Figure 3. For each map template $m_k \in M$, C_k and all generated template graphs q_i are evaluated based on the current state of the execution context X, which contains a reference to the currently processed map template m_k (`activeTmpl`) in order to retrieve w_k and

s_k. The base workbook w_k and the base worksheet s_k are required for the evaluation of relative range references and obtaining values from the cells of the spreadsheets. The execution context also contains a reference to the target graph (`targetGraph`) where the generated statements are inserted. While in Section 4 we used $[[G_k]]$ to denote the evaluation of G_k including the application of transform operations F_k, the notation of $[[C_k]]_X$ and $[[q_i]]_X$ in the flow chart only represents the evaluation of XLWrap expressions for the given graphs C_k and q_i. The application of F_k is completely hidden by the `TemplateModelGenerator`, which subsequently applies the defined transform operations $f_l \in F_k$ against G_k and returns multiple template graphs q_i.

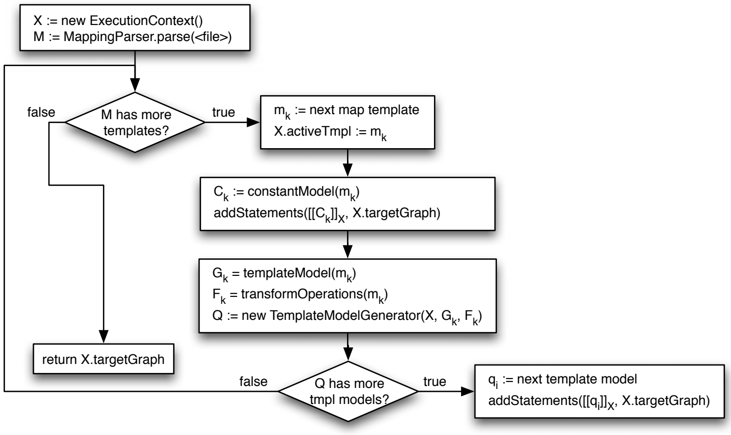

X := new ExecutionContext()
M := MappingParser.parse(<file>)
M has more templates?
false
true
m_k := next map template
X.activeTmpl := m_k
C_k := constantModel(m_k)
addStatements([[C_k]]_X, X.targetGraph)
G_k = templateModel(m_k)
F_k = transformOperations(m_k)
Q := new TemplateModelGenerator(X, G_k, F_k)
return X.targetGraph
false
Q has more tmpl models?
true
q_i := next template model
addStatements([[q_i]]_X, X.targetGraph)

Fig. 3. Overview of the wrapping process

The `TemplateModelGenerator` is implemented as an iterator which uses a sequence of stacked instances of `TransformationStage` – one for each transform operation f_l. Each stage transforms its current *stage graph* $q^{f_l}_{i_1,\ldots,i_n}$ according to the corresponding transform operation as depicted in Figure 4. Initially, all stage graphs are equal to the template graph: $q^{f_l}_{0,\ldots,0} = G_k$. The blue nodes on the bottom represent final template graphs which are returned by `TemplateModelGenerator`. Each call to `TemplateModelGenerator.hasNext()` results in a transformation at the lowest stage that has more transformations to apply. For example, if there are no more rows to shift by f_1, the corresponding stage triggers its parent stage and tracks back its internal state. Likewise, if the condition defined for the transform operation does not hold, it is skipped and the parent stage is triggered.

When a template graph is applied, before its triples are added into the target graph, any blank node with a `xl:uri` property is replaced with a URI node, blank node labels with equal `xl:id` properties are aligned, and any `xl:Expr` literal is evaluated as an XLWrap expression e. The result of $[[e]]$ is an instance of `XLExprValue`, which can be a URI, blank node, string, long integer, double, boolean, or date value. When obtaining cell values, the type is automatically detected based on the type annotation (Definition 4). When creating literals for the target graph, long integers and floats are

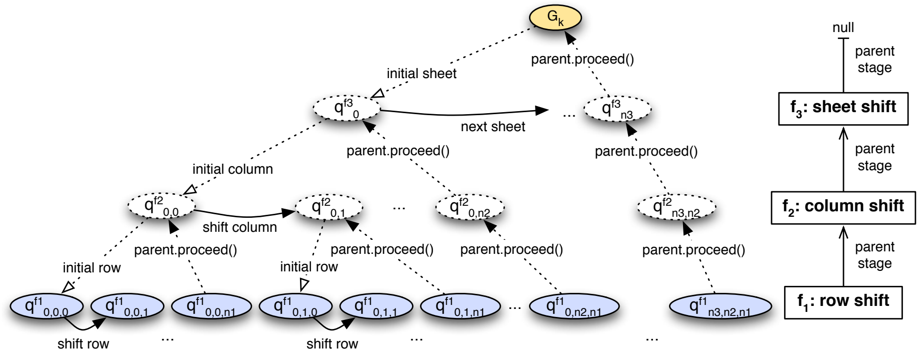


Fig. 4. Transform stages for the mapping specification of Listing 2

automatically reduced to the required size as long as they have not been explicitly casted with a type casting function before. Depending on the type, a matching typed literal is created.

For the example spreadsheet given in Table 1, after applying the mapping in Listing 2, the following triples are generated:

```
ex:revenue_Austria_2007_Product1 a ex:Revenue ;
      ex:country <http://dbpedia.org/resource/Austria> ;
      ex:itemsSold "342"^^<http://www.w3.org/2001/XMLSchema#short> ;
      ex:product "Product1" ;
      ex:revenue "7866"^^<http://www.w3.org/2001/XMLSchema#int> ;
      ex:year <http://dbpedia.org/resource/2007> .
ex:revenue_Austria_2007_Product2 a ex:Revenue ;
      ex:country <http://dbpedia.org/resource/Austria> ;
      ex:itemsSold "4333"^^<http://www.w3.org/2001/XMLSchema#short> ;
      ex:product "Product2" ;
      ex:revenue "1005256"^^<http://www.w3.org/2001/XMLSchema#int> ;
      ex:year <http://dbpedia.org/resource/2007> .
ex:revenue_Austria_2007_Product3 ...
ex:revenue_Austria_2007_Product4 ...
ex:revenue_Austria_2008_Product1 ...
...
ex:revenue_Austria_2009_Product1 ...
...
ex:revenue_Germany_2007_Product1 ...
...
ex:revenue_Germany_2009_Product1 ...
...
ex:revenue_Germany_2009_Product5 ...
```

Range reference sub-expressions of the stage template graph $q^{f1}_{0,0,0} = G_k$ are shifted down by one row first until the condition `LEFT(A4, 7) == 'Product'` is false, producing resources for all products sold in Austria in 2007. However, only those range references within the range restriction z_{f1} = `"A4; B4:C4"` are actually transformed. For instance, the expression `"A4"` (literal for `ex:product`) is subsumed by the restriction range and is therefore changed to `"A5"`, but `"DBP_YEAR(B2)"` remains unchanged. Next, $q^{f2}_{0,1}$ is calculated by a 2-step column shift of $q^{f2}_{0,0}$. The stage model of the sub-stage is initialized as $q^{f1}_{0,1,0} := q^{f2}_{0,1}$ for the next execution of f_1 (row shift). If both, f_1 and f_2

have no more transformations (or both stage conditions do not hold), the sheet is shifted according to f_3, producing similar RDF data for Germany.

Transform operations are not only applied to range references in `xl:Expr` literals of $q^{f_l}_{i_1,\ldots,i_n}$, they must be applied also to the range restrictions z^{f_l} and to the conditions $e^{f_l}_c$ of the corresponding transform operations. For instance, the range restriction on the row shift `"A4; B4:C4"` has to be shifted to `"A5; B5:C5"` in the first stage and then to `"A4; D4:E4"`, `"A5; D5:E5"`, and `"A4; F4:G4"`, `"A5; F5:G5"`, etc. in the second stage. When proceeding at the second stage, the transformation of the original f_1-restriction is itself restricted by the current range restriction of f_2, which is `"B2; B4:C4"`. As visualized in Figure 5, thus only a subset of `"A4; B4:C4"` is shifted leading to `"A4; D4:E4"`. Currently, XLWrap is not capable of automatically splitting arbitrary box ranges based on restrict ranges. This is why, "A4; B4:C4" was not specified as "A4:C4" in the mapping[9]. However, intersections of box ranges are detected during the initialization of a map template in order to be corrected.

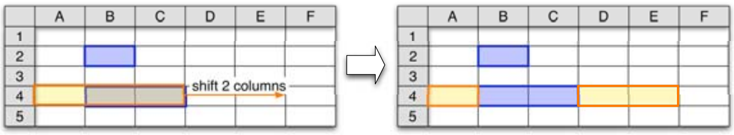


Fig. 5. Column shift of range restriction "A4; B4:C4" restricted by "B2; B4:C4"

6 Conclusion

In this contribution we have presented XLWrap, which is an RDF-centric mapping approach to support the transformation of spreadsheets with different representation models to arbitrary RDF graphs. The mapping concept has been formally defined and implemented based on the Jena Semantic Web framework. The server component called XLWrap-Server was not further discussed due to the page limit. It is a stand-alone Web application based on Joseki and Snorql from the D2R-Server project including a SPARQL endpoint and a linked data interface.

XLWrap is powerful enough to represent mappings for spreadsheets with different representation models and target graphs. Because it supports external references, HTTP URLs, and the wrapping of multiple spreadsheets into a combined cache including OWL inference, it can be used very easily to semantically integrate multiple spreadsheets locally or in intranets and extranets. The possibility of adding custom functions – which is a matter of extending `XLExprFunction` and providing an implementation for `eval()` – can be very practical for end-users. Beside adding custom mathematical and statistical functions, it is possible to access a database or Web resources by XLWrap functions. The future support for aggregate functions in SPARQL is a very important requirement in order to support typical operations on spreadsheet data.

[9] Especially in combination with the *multi sheet* and *any range*, ranges cannot be split straightforward and the implementation would additionally require support for exclusion ranges.

Future work will include the development of a graphical support tool including some kind of mapping debugger and auto-detection of cross-tables to facilitate the mapping specification task. Considerable work regarding auto-detection of headers and units has already been published [1,2].

Acknowledgements

This work is funded by the Austrian BMBWK (Federal Ministry for Education, Science and Culture), contract GZ BMWF-10.220/0002-II/10/2007. We would like to thank Bernhard Haslhofer, Bernhard Schandl, Thomas Leitner, and Berndt Zinnöcker for the helpful inputs and discussions held during the implementation of XLWrap.

References

1. Abraham, R., Erwig, M.: Header and unit inference for spreadsheets through spatial analyses. In: VLHCC 2004: Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing, Washington, DC, USA, pp. 165–172. IEEE Computer Society Press, Los Alamitos (2004)
2. Chambers, C., Erwig, M.: Dimension inference in spreadsheets. In: VLHCC 2008: Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, Washington, DC, USA, pp. 123–130. IEEE Computer Society Press, Los Alamitos (2008)
3. Cyganiak, R., Bizer, C.: D2R Server – Publishing Relational Databases on the Web as SPARQL Endpoints. In: Developers Track at the 15th International World Wide Web Conference (WWW2006), Edinburgh, Scotland (May 2006)
4. Group, M., Reck, R.P.: Excel2RDF, `http://www.mindswap.org/~rreck/excel2rdf.shtml` (Last visit, June 2009)
5. Han, L., Finin, T.W., Parr, C.S., Sachs, J., Joshi, A.: RDF123: From Spreadsheets to RDF. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 451–466. Springer, Heidelberg (2008)
6. HP Labs, Bristol, UK: Joseki – A SPARQL Server for Jena, `http://www.joseki.org/` (Last visit, June 2009)
7. Langegger, A., Wöß, W.: SemWIQ – Semantic Web Integrator and Query Engine. In: Hegering, H.G., Lehmann, A., Ohlbach, H.J., Scheideler, C. (eds.) Beiträge der 38. Jahrestagung der Gesellschaft für Informatik e.V (GI), vol. 1. Bonner Köllen Verlag (2008)
8. Maurizio, L.: Data integration: a theoretical perspective. In: PODS 2002: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 233–246. ACM, New York (2002)
9. Valente, A., Van brackle, D., Chalupsky, H., Edwards, G.: Implementing logic spreadsheets in less. Knowl. Eng. Rev. 22(3), 237–253 (2007)