M^2: a Language for Mapping Spreadsheets to OWL

Martin J. O'Connor[1], Christian Halaschek-Wiener[2], Mark Musen[1]

[1]Stanford Center for Biomedical Informatics Research
Stanford, CA 94305
[2]Clados Management, LLC
San Mateo, CA 94401

In this paper, we describe a mapping language for converting data contained in arbitrary spreadsheets into the Web Ontology Language (OWL). The developed language overcomes shortcomings with existing spreadsheet mapping techniques, including their restriction to well-formed spreadsheets reminiscent of a single relational database table and verbose syntax for expressing mapping rules when transforming spreadsheet contents into OWL. We additionally present an implementation of the mapping approach, Mapping Master, which is available as a plug-in for the Protégé ontology editor.

1 Introduction

One of hurdles that new and existing users of Semantic Web standards continue to face is converting preexisting, non-Semantic Web encoded information into one of the many Semantic Web languages (e.g., RDF, OWL). In some domains, a large deal of information is represented in spreadsheets (e.g., financial services). This volume of spreadsheet content has motivated both academia [1] and industry [2, 8] to develop a variety of general-purpose spreadsheet mapping techniques to avoid manually encoding spreadsheet content in OWL or writing custom extraction programs.

However, existing mapping approaches suffer from a variety of limitations. First, many mapping techniques assume very simple data models within spreadsheets [3]. Typically, it is assumed that each table in a spreadsheet adheres to a relational model where each row in the table describes a different entity and each column describes an attribute for that entity; we refer to this as the 'entity-per-row' assumption. Unfortunately, there are numerous real-world spreadsheets that do not adhere to this simple data model, as many spreadsheet-authoring tools are extremely flexible and do not restrict the manner in which users author tabular structures. Common examples of complex layouts can be found in the financial domain. Here, analysts or companies publish sales forecasts or results, which are typically represented by tables that have products or market segments listed in a column, quarters or years listed in a row, and sales figures specified for each product/market segment and date. An example of this type of spreadsheet is illustrated below:

<>	A	B	C	D	E
1	**Revenues—Major Pharmaceutical Products**				
2	(Millions of Dollars)		YEAR ENDED DECEMBER 31,		
3	PRODUCT	PRIMARY INDICATION	2008	2007	2006
4	**Infectious and respitory diseases**				
5	Zyvox	Bacterial infections	1,115	944	782
6	Vfend	Fungal infections	743	632	515
7	Zithromax/Zmax	Bacterial infections	429	438	638
8	Difulcan	Fungal infections	373	415	435
9	...				

Figure 1: Drugs, their primary use, and their sales for a number of years.[1]

Recently, there have been efforts to overcome this limitation and support mappings for arbitrary spreadsheets [1]. However, to the best of our knowledge, these approaches use a RDF triples-based approach to encode mapping rules. They can be effective when mapping spreadsheet content to RDF but are very cumbersome when encoding that content in OWL due to its verbose RDF serialization. To illustrate, let's assume a financial analyst wants to model the information in Figure 1 in OWL. First, assume the analyst models each drug as a class that has OWL property restrictions for the drug's treated disease type and primary indication[2]. Using this representation, the drug `Zyvox` could be modeled as follows (presented using the Manchester Syntax [4]):

Class: `Zyvox` SubClassOf: `Drug` and `treatsDisease` some '`Infectious and respiratory diseases`' and `forIndication` some '`Bacterial infections`' (Ex. 1)

Next, assume the analyst models each sales figure as an OWL class that has OWL property restrictions for the drug, date, and actual amount. Thus, the cell C5 could be modeled using a new OWL class, `sales1`, as follows:

Class: `sales1` SubClassOf: `SalesAmount` and `forDrug` some `Zyvox` and `forDate` has 2008 and `amount` has "1,115" (Ex. 2)

To encode the OWL class axioms in Ex. 1 & 2 in RDF, 10s of triples are required because the RDF serializations for `owl:intersectionOf`, `owl:hasValue` and `owl:someValuesFrom` require multiple triples. Therefore, using state-of-the-art mapping techniques, even simple mapping rules can be extremely verbose.

To overcome these limitations, we propose a new declarative OWL-centric mapping language that supports arbitrary spreadsheet-to-OWL mappings. The language also supports syntactic transformations of cell contents, as well as inline OWL axioms involving classes, properties and individuals extracted from cell contents. In the end, the mapping language enables mapping information from arbitrary spreadsheets to OWL using a compact, user-friendly syntax.

[1] Source: Pfizer 2008 Financial Report: http://media.pfizer.com/files/annualreport/2008/financial/financial2008.pdf

[2] A philosophical discussion regarding whether this information should be modeled as classes or individuals is out of the scope of this paper. However, a class-based representation is consistent with modeling conventions used in widely accepted biomedical ontologies.

2 Background and Related Work

A variety of systems have been developed to map spreadsheet content to RDF. The earliest systems include Excel2RDF [6] and Convert2RDF [6]. Both systems support basic mappings from entity-per-row spreadsheets. The later RDF123 system [3] supports less restricted data models. However, it is still primarily row-centric. While complex mapping conditions are supported, the mapping language still fundamentally assumes entity-per-row storage. The recent XLWrap system [1] attempts to addresses this shortcoming. It allows data to be organized in essentially arbitrary ways and supports an expressive mapping language for generating RDF content. Other primarily metadata-based systems include MIT's Simile project, Aperature/Nepomuk from Semantic Discovery Systems, and Cambridge Semantic's Anzo for Excel [8].

Some systems use an XSLT-based approach to map automatically-generated XML representations of spreadsheets to RDF. However, these approaches can be very cumbersome and are generally useful for only a small range of simple mappings. A related, higher level approach is to use importation tools to generate OWL or RDF tabular representations of spreadsheet data and to then map these tabular representations to domain ontologies using rule or scripting languages. For example, TopBraid Composer's SPARQLMotion [2] provides a range of scripting modules for generating RDF from tabular data imported from spreadsheets. The authors have used a similar approach with a data importation tool called DataMaster [9] that uses SWRL [10] rules to map spreadsheet data imported by DataMaster to domain-level constructs. While these approaches provide great flexibility, a multitude of rules or mapping scripts can quickly accumulate, which can be difficult to manage and debug.

A general shortcoming of existing mapping systems is that they are RDF-centric and are not designed to directly work with OWL. The only exception known to the authors is ExcelImport [11]. However, this tool assumes simple-entity-per row spreadsheets and provides only a small set of OWL constructs that are specified graphically. It does not support a mapping language.

3 Mapping Language

As discussed previously, the primary goal is to address the limitations of existing mapping tools by developing a new declarative OWL-centric mapping language. Importantly, this domain-specific language (DSL) should support complex and arbitrary spreadsheets that do not conform to the entity-per-row assumption. To ensure the mapping approach is compatible with the workflow familiar to users of spreadsheet tools, the language must also allow mappings of data spread over multiple sheets. A related requirement is that is should support mapping of data that may be distributed non-uniformly in individual sheets (for example, multiple disconnected tables in a sheet representing the same underlying information). Additionally, it should allow the selective extraction of data from within cells.

Full coverage of all OWL constructs is also a primary goal of the mapping language. In addition to supporting the definition of simple OWL entities such as named classes, properties, and individuals, class expressions and potentially complex necessary and sufficient declarations should be expressible. While an RDF triple-based mapping mechanism can in principle generate arbitrary OWL constructs, such an approach is not practical in general because of OWL's complex RDF serialization. This approach would also not be compatible with the goal of producing a concise language.

Additionally, the language should not only be concise but also simple to learn for users familiar with both OWL and spreadsheet tools. An additional usability difficulty when developing custom languages is providing debugging support. The typical levels of complexity when mapping from spreadsheets to OWL make this support particularly crucial. An important language design goal is thus to support instantaneous preview of mapping results before they are executed and to allow those previews to be updated dynamically when underlying data are changed.

3.1 Core M^2 Language

Rather than design a DSL from scratch, the proposed language is built upon the Manchester Syntax [4], a widely-used DSL for declaratively describing OWL ontologies. As illustrated in Ex. 1 & 2, this DSL has concise clauses for defining common OWL entities. It also provides full coverage of all OWL constructs and is familiar to most users of OWL since it is the standard presentation syntax used by the Protégé ontology editing tools. It has a very clean language definition, allowing it to be extended in a principled way. The DSL that we have defined—called M^2, or *mapping master*—is a superset the Manchester Syntax, so any valid Manchester Syntax expression is also a valid M^2 expression[3].

M^2 extends the Manchester Syntax to allow references to spreadsheet content in expressions. It introduces a new *reference* clause to support these references. This clause can indicate one or more cells in spreadsheet. In this DSL, any clause in a Manchester Syntax expression that indicates an OWL class, OWL property, OWL individual, data type, or data value can be substituted with this reference clause.

References clauses are prefixed with the character '@' and are followed by an Excel-style cell reference. In the standard Excel cell notation, cells extend from A1 in the top left corner of a sheet within a spreadsheet to successively higher columns and rows, with alpha characters referring to columns and numerical values referring to rows. For example, a reference to cell A5 in a spreadsheet is written as follows:

@A5

The above cell specification indicates that the reference is relative, meaning that if a formula containing the reference is moved then the row and column components of

[3] A full description of the language can be found on the MappingMaster wiki [11]. Its BNF can be found at: http://swrl.stanford.edu/MappingMaster/1.0/BNF/MappingMasterParser.html

the reference are updated appropriately. An equivalent absolute reference, again adopting Excel notation, can be written as follows:

@A5

References can also be preceded by a sheet name. For example, a reference to the same cell in the sheet "Sales Data" can be written:

@"Sales Data"!A5

In many real-world spreadsheets, users may wish to evaluate the same mapping formula over a range of spreadsheet cells. For example, an analyst would likely want to evaluate the mapping expression in Ex. 2 over the cell range C5:E8 of the spreadsheet presented in Figure 1. To avoid repeatedly defining mapping expressions for each cell such a range, M2 allows the user to define a cell range and then use wildcards, denoted by '*', in place of row and/or column references (illustrated in Figure 2). Then when the mapping expression is evaluated, the cell range is iterated over and the wildcards are replaced with the current row/column.

This reference clause can then be used in M^2 expressions to define OWL constructs using spreadsheet content. For example, an M^2 expression to take the name in cell A5 of the spreadsheet in Figure 1 and declare an OWL named class that is a subclass of an existing `Drug` class can be written:

Class: @A5 SubClassOf: Drug

This expression declares an OWL class named by the contents of cell A5 ('Zyvox' in this case) and asserts that it is a subclass of class `Drug`. If the class has previously been declared and it is not already a subclass of `Drug` then that relationship will be asserted.

Using this approach, any OWL axiom can be declared using the appropriate Manchester Syntax clause, with references used in these clauses to specify spreadsheet content. For example, an M^2 expression to instead declare an individual of type `Drug` using the contents cell A5 as its name can be written:

Individual: @A5 Types: Drug

Ambiguities may arise regarding the type of the entity being referenced. In the above drug class declaration example, it is clear that @A5 refers to an OWL class. However, the type can not always be inferred. Explicit entity type specifications are provided to deal with this case. To support this specification, a reference may be optionally followed by a parenthesis-enclosed *entity type specification* to explicitly declare the type of referenced entity. This specification can indicate that the entity is an OWL named class, an OWL object or data property, or an OWL individual or a data type. The M^2 keywords to specify the types are: `Class`, `ObjectProperty`, `DataProperty`, `Individual`, and any XSD type name (e.g., `xsd:int`). Using this specification, the above drug declaration, for example, can be written:

Class: @A5(Class) SubClassOf: Drug

In many cases, specifying the super class, super property, individual class membership, or the data type of referenced entities is also desired. While these types of relationships can be defined using standard Manchester Syntax expressions, this

approach will often entail the use of multiple mapping expressions. To concisely support defining these types of relationships, a reference may optionally be followed by a parenthesis-enclosed list of type names. Using this approach, the above drug declaration, for example, can be written as follows:

Class: @A5(Drug)

These type specifications can themselves be cell references and can be nested to arbitrary depths, though excessive use of nesting may make expressions difficult to understand and debug. Super properties, individual class membership, and data types can be specified in the same way.

A variety of name encoding strategies are supported when creating entities from spreadsheet content. The primary strategies are to either use direct URI-based names (equivalent to using `rdf:about` or `rdf:ID` clauses in an RDF serialization of OWL) or to use `rdfs:label` annotation values. The default naming encoding uses the `rdfs:label` annotation property. The default may also be changed globally (discussed in Section 5). Using `rdfs:label` encoding, the OWL entity generated from a cell referenced is given an automatically generated (and non meaningful) URI and its `rdfs:label` annotation value is set to the content of the cell.

A *name encoding clause* is provided to explicitly specify a desired encoding. As with entity type specifications, this clause is contained in parenthesis after the cell reference. The M^2 keywords to specify the three types of encoding are `rdf:about`, `rdf:ID`, and `rdfs:label`. Using this clause, a specification of `rdf:ID` encoding for the previous drug example can be written:

Class: @A5(rdf:ID Drug)

The default M^2 behavior is to directly use the contents of the referenced cell when encoding a name. However, this default can be overridden using an optional *value specification clause*. This clause is indicated by the '=' character immediately after the encoding specification keyword and is followed by a parenthesis-enclosed, comma-separated list of *value specifications*, which are appended to each other. These value specifications can be cell references or values. For example, an expression that extends the earlier reference to specify that the entity created from cell A5 is to use `rdfs:label` name encoding and that the name is to be the value of the cell preceded by the string "Sale:" can be written as follows:

Class: @A5(rdfs:label=("Sale:", @A5) Drug)

Value specification references are not restricted to the referenced cell itself and may indicate arbitrary cells. More than one encoding can also be specified for a particular reference so, for example, names and annotation values can be generated for a particular entity using the contents of different cells.

A similar approach can be used to selectively extract values from referenced cells. A *regular expression capture group clause* is provided and can be used in any position in a value specification clause. This clause is contained in a quoted string enclosed by square parenthesis. For example, if cell A5 in the previous example contained the

string “Pfizer:Zyvox” but only the text following the ‘:’ character is to be used in the label encoding, an appropriate capture expression could be written as:

Class: @A5(Drug rdfs:label=(“Sale:”, [“:([a-zA-Z][a-zA-Z0-9])”]))*

Note that parentheses around the sub-expressions in a regular expression clause specify capture groups and indicate that the matched strings are to be extracted. In some cases, more than one capture group may be matched for a cell value, in which case they are extracted in the order that they are matched and appended to each other.

To deal with missing cell values, default values can also be specified in references. A *default value clause* is provided to assign these values. This clause is indicated by the keyword `mm:default` and is separated from a value specification clause by a comma. It is followed by a parenthesis-enclosed, comma-separated list of value specifications. For example, the following expression uses this clause to indicate that the value “Unknown” should be used as the created class label if cell A5 is empty:

Class: @A5(rdfs:label, mm:default=(“Unknown”) Drug)

Each value specification clause can be followed by a default, so different default values can be used for `rdf:ID` and `rdfs:label` encodings. This clause can also be used for data values and may itself contain M^2 references.

Finally, a *filter clause* may be used to indicate that cells that do not meet particular criteria should be ignored. This clause is indicated by the keyword `mm:filter` and, like value and type specifications, is enclosed in parenthesis after a cell specification. This keyword is followed by the ‘=’ character and a quoted condition, which is specified using Excel-style Boolean condition notation. Using this clause, a variant of the previous expression that skips cells with the value ‘Zyvox’ can be written:

Class: @A5(mm:filter=“A5<>Zyvox” Drug)

Excel-style conditions are very expressive and allow almost arbitrary filter criteria.

3.2 M^2 Mapping Process

The M^2 mapping process takes a source spreadsheet, a set of M^2 expressions, and a target ontology. The mappings are processed in three phases. In first phase, every expression is preprocessed and the relevant content specified by references in these expressions is retrieved from the source spreadsheet. This content, which will either specify a data value or the name of a data type or an OWL entity, is substituted for each reference in an M^2 expression to generate a valid Manchester Syntax expression. The second phase declares all referenced OWL entities that are not already declared in the target ontology. The type specification for each reference is used to generate the appropriate declaration clause. Any super class, super property, individual class membership, or data type specifications in the reference are also declared in this phase. Once the entities have been declared, the third phase involves sending the final Manchester Syntax expression to a Manchester Syntax processor. This processor will populate the target ontology with the OWL axioms specified by the expressions.

At the end of phase one, the generated expressions can be checked for syntactic correctness. They can also be previewed at this stage if desired, allowing users to see the final entity names expanded within their enclosing M^2 expression.

M^2 supports several preprocessing directives to specify configuration options for the mapping process. These directives include the ability to declare both a default namespace for generated entities and to specify prefix-to-namespace mappings. The latter option allows M^2 to deal with cells that contain both prefixed and fully qualified URI entity names. An option is also supported to indicate that cell values refer to OWL entities using annotation values. In the default case, these names—be they prefixed, fully qualified, or annotated—are assumed to either refer to existing OWL entities or to name entities that are to be declared during the import process. M^2 supports a pair of options to modify this behavior. The first option can be set to indicate that an error should be thrown if a name refers to an existing entity in the target ontology; the second option indicates that an error should be thrown if the name foes not refer to an existing entity. A related option deals with the possible ambiguity introduced by the use of annotation value references. It can be set to produce an error if more than one existing OWL entity could be named by the value.

A final set of options is designed for dealing with missing values. Three behaviors can be specified: (1) throw an error if any referenced cell contains a missing value; (2) skip the affected mapping expression for a missing-value cell and continue to other cells; and (3) attempt to generate an expression by conservatively dropping only the sub-expressions that reference the missing-value cell.

M^2 provides an *option specification clause* for each option type. The general form of this option specification clause is a keyword followed by a value. For example, the default name encoding for all mappings can be written:

mm:DefaultNameEncoding = rdfs:label

It is noted that OWL axioms generated during the mapping process may cause inconsistencies in the target ontology. Further, since users have full control over M^2 expression authoring, the expressions can also generate axioms that are inconsistent with each other. To immediately detect such inconsistencies, an ideal implementation would invoke an incremental reasoner after each expression is executed.

4 Implementation

A parser, editor, and execution engine for the M^2 DSL have been written. The parser currently supports core Manchester Syntax OWL entity declarations plus arbitrary class expressions, though full coverage is anticipated soon. Additionally, a development environment has been released as an open source plugin to the Protégé-OWL editor [5]. This development environment includes Java APIs for interacting with M^2 from software applications and a graphical user interface. This user interface is available as a Protégé-OWL plug-in called Mapping Master and provides an editor for defining, managing and executing M^2 expressions. It supports the loading and

previewing of spreadsheets defined in both Excel and CSV formats. An interface to interactively specify the array of configuration options supported by the M^2 DSL is also provided. M^2 expressions can also be defined interactively and then executed to map the contents of these loaded spreadsheets to a target ontology (see Figure 2). The plug-in includes a persistence mechanism to save and reload these mappings.

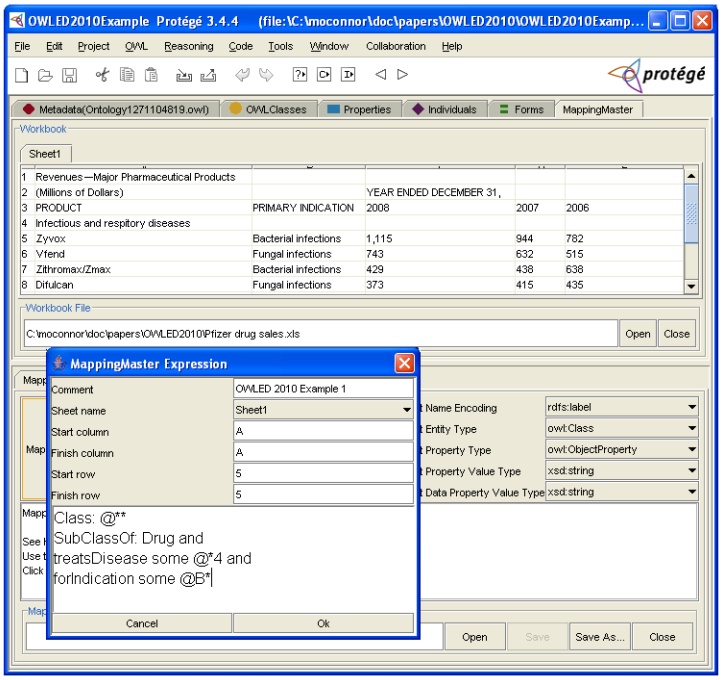

Figure 2. Screen shot of the Mapping Master Protégé plug-in showing its spreadsheet preview screen, configuration control panel, and M^2 expression editor.

5 Conclusion

Recent approaches on mapping information contained in spreadsheets to OWL suffer from a variety of limitations, including assuming well-formed spreadsheets reminiscent of a single relational database table and verbose syntaxes for expressing mapping rules. In this paper, we have overcome these limitations by developing a mapping language, M^2, which is based on an extension of the OWL Manchester Syntax that supports arbitrary spreadsheet cell references. This mapping language provides a compact, user-friendly approach for expressing mapping rules for arbitrary spreadsheets. The language also supports syntactic transformations of cell contents, as well as inline OWL axioms involving classes, properties and individuals extracted from cell contents. Lastly, we have recently released a free, open source implementation of the approach as a Protégé plug-in called Mapping Master.

The plug-in has been used successfully by several research groups. For example, researchers from the OBI consortium [12] have used it to import analyte assay instances from spread sheets and encode them in OWL. Internally, it has been used by colleagues to import clinical definition from spreadsheets for a biomedical terminology project. We have extensively tested the expressivity of the language against a range of publicly available financial spreadsheets. We are currently in the process of carrying out an in-depth evaluation of the language and tool.

Future work includes extending the mapping approach to work directly within Microsoft Excel, which will allow mapping expressions to be authored directly in cells and use native Excel cell references and functions. This will additionally enable standard Excel formula operations, such as copy and paste, for mapping expressions associated with cells, as well as allow interactive previews of M^2 expressions in cells using references substituted with cell values. Other potential future work includes supporting user-defined functions in mapping expressions.

Acknowledgments

This research was supported in part by STTR Award #0750543 from the National Science Foundation. We would also like to thank Peter Moore and Natasha Noy.

References

[1] Langegger, A., Woss, W. XLWrap-Querying and Integrating Arbitrary Spreadsheets with SPARQL. In: ISWC 2009, LNCS 5823, Springer (2008).
[2] TopBraid Composer: http://www.topbraidcomposer.com
[3] Han, L., Finin, T.W., Parr, C.S., Sachs, J., Joshi, A. RDF123: From Spreadsheets to RDF. In: ISWC 2008. LNCS 5318, pp. 451–466, Springer (2008).
[4] Manchester OWL Syntax: http://www.w3.org/TR/owl2-manchester-syntax/
[5] Mapping Master: http://protege.cim3.net/cgi-bin/wiki.pl?MappingMaster
[6] Reck, R.P.: Excel2RDF for Microsoft Windows, http://www.mindswap.org/rreck/~excel2rdf.shtml
[7] Grove, M. Mindswap Convert2RDF Tool, http://www.mindawap.org/~mhgoeve/convert/
[8] Huynh, D., Karger, D., Miller, R. Exhibit: lightweight structured data publishing. In: Proceedings of the 16th International Conference on World Wide Web (2007).
[9] O'Connor, M.J., Shankar, R.D., Tu, S.W., Nyulas, C.I., Das, A.K. Developing a Web-Based Application using OWL and SWRL. AAAI Spring Symposium, Stanford, CA, USA (2008).
[10] SWRL Submission: http://www.w3.org/Submission/SWRL
[11] ExcelImport: http://code.google.com/p/co-ode-owl-plugins/wiki/ExcelImport
[12]OBI Consortium: http://obi-ontology.org/page/Consortium