Mapping Master: a Flexible Approach for Mapping Spreadsheets to OWL

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

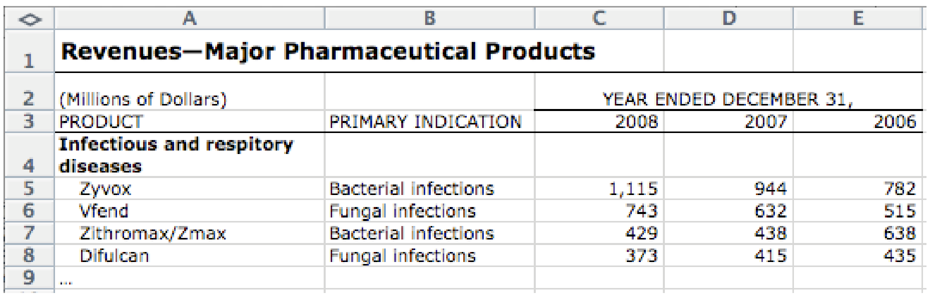
[1]Stanford Center for Biomedical Informatics Research
Stanford, CA 94305, USA
[2]Clados Management, LLC
San Mateo, CA 94401, USA

Abstract. We describe a mapping language for converting data contained in spreadsheets into the Web Ontology Language (OWL). The developed language, called M^2, overcomes shortcomings with existing 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. The M^2 language provides expressive, yet concise mechanisms to create both individual and class axioms when generating OWL ontologies. 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 the 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 this information is represented in spreadsheets (e.g., financial services), which 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.

Existing mapping approaches, however, 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 in Figure 1.



	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 the entity-per-row limitation and to support mappings for arbitrary spreadsheets [1]. However, to the best of our knowledge, these approaches use an RDF triples-based approach to encode mapping rules. They can be effective when mapping spreadsheet content to RDF, but are very cumbersome when encoding 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, dozens of triples are required because the RDF serializations for `owl:intersectionOf`, `owl:hasValue` and `owl:someValuesFrom` require multiple triples. Therefore, using currently available 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 complex 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 Related Work

A variety of systems have been developed to map spreadsheet content to RDF. The earliest approach include Excel2RDF [6] and Convert2RDF [7]. Both systems provide basic mapping languages to support mappings from entity-per-row spreadsheets to RDF. The later RDF123 approach [3] has a mapping language that allows complex mapping conditions that support less restricted spreadsheet data models but the language still fundamentally assumes entity-per-row storage. The recent XLWrap mapping approach [1] attempts to address this shortcoming. It allows data to be organized in essentially arbitrary ways and provides an expressive mapping language for generating RDF content. However, the resulting mapping language is rather verbose. Other spreadsheet mapping systems include MIT's Simile project and Cambridge Semantic's Anzo for Excel [8], though these systems are primarily based on metadata.

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 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 additionally does not support a mapping language.

It is lastly noted that techniques have also been developed for mapping information stored in relational database management systems to RDF and OWL [13]. While possibly applicable to simple spreadsheets adhering to the entity-per-row assumption, such techniques are not suited for mapping semi-structured tables such as that presented in Figure 1.

3 Mapping Language

The primary goal of this work 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 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 it 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 always practical because of OWL's complex RDF serialization. This approach would also conflict 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. A general usability issue when developing a custom language is providing debugging support for that language. The typical levels of complexity when mapping from spreadsheets to OWL makes this support crucial. In particular, the ability to preview the final result of a mapping expression before executing it can greatly assist in debugging. 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 the underlying data are changed.

3.1 Core M^2 Language

Rather than designing 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 [14, 15]. 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 of the Manchester Syntax, so any valid Manchester Syntax expression is also a valid M^2 expression. In the remainder of this section we provide an overview of the M^2 language. We refer the interested reader to the M^2 wiki [5] for a full description of the language. Additionally, its BNF is available at [17].

M^2 Reference Clause. M^2 extends the Manchester Syntax to allow references to spreadsheet content in expressions. It introduces a new *reference* clause to support these references (see Figure 2). This clause indicates one or more cells in spreadsheet. In the 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[3]. 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 copied to another cell then the row and column components of the reference are updated appropriately. An equivalent absolute reference, again adopting Excel notation, can be written as follows:

@A5

```
reference ::= '@' cell-ref [ '(' [ entity-type ]
                                 { value-encoding }
                                 [ shift-setting ]
                                 { empty-value-handling }
                                 [ default-value ]
                                 [ filter ]
                                 { defining-type } ')' ]
cell-ref ::= [ a valid Excel sheet reference '!' ] column-ref row-ref
column-ref ::= '*' | a valid Excel column reference
row-ref ::= '*' | a valid Excel row reference
entity-type :: = 'Class' | 'ObjectProperty' | 'DataProperty' |
                 'Individual' | xsd-type
xsd-type ::= 'xsd:int' | 'xsd:float' | ...
value-encoding ::= ( 'rdfs:label' | 'rdf:ID' ) [ value-specification ]
value-specification ::= '=' '(' value-item { ',' value-item } ')'
value-item ::= reference | literal | capture-expression
capture-expression ::= '[' a valid Java pattern expression ']'
default-value ::= 'mm:default' '=' '(' value-item { ',' value-item } ')'
empty-value-handling ::= empty-location-handling | empty-ID-handling |
                         empty-label-handling
empty-location-handling ::= 'mm:ErrorIfEmptyLocation' | ...
shift-setting ::= 'mm:NoShift' | 'mm:ShiftLeft' | 'mm:ShiftRight' | ...
filter ::= 'mm:default' '=' '"' a valid Excel Boolean expression '"'
defining-type ::= reference | classExpression | dataPropertyExpression |
                  objectPropertyExpression | xsd-type
```

Figure 2. Partial BNF of M^2 reference clause.

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 want 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 in such a range, M^2 allows the user to define a cell range and then use wildcards, denoted by '*', in place of row and/or column references (defined in Figure 2). Then, when the mapping expression is evaluated, the mapper iterates over the cell range and the wildcards are replaced with the current row and column.

[3] A formalization of spreadsheets is omitted here due to their widespread use and adoption.

The reference clause can be used in M^2 expressions to define OWL constructs using spreadsheet content. For example, an M^2 expression can easily be defined 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 as follows:

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 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, a 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

M^2 Mapping Directives. The M^2 language additionally extends the Manchester syntax with a variety of directives, which facilitate the mapping process and help achieve the goals previously described. In this section, we discuss a variety of these directives and illustrate their use.

In the above drug class declaration example, it is clear that @A5 refers to an OWL class. However, the type cannot always be inferred and ambiguities may arise regarding the type of the entity being referenced. To deal with this case, explicit entity type specifications are provided. Specifically, 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 a named OWL 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 enclosed by parentheses 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.

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 parentheses 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)

M^2 Missing Value Handling. 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 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)

Additional behaviors are also supported to deal with missing cell values. M^2's default behavior is to skip an entire expression if it contains any references with empty cells. Four keywords are supplied to modify this behavior. These keywords indicate that: (1) an error should be thrown if a cell value is missing and the mapping process should be stopped (`mm:ErrorIfEmptyLocation`); (2) expressions containing references with empty cells should be skipped (`mm:SkipIfEmptyLocation`); (3) expressions containing references with empty cells should generate a warning in addition to being skipped (`mm:WarningIfEmptyLocation`); and (4) expressions containing such empty cells should be processed (`mm:ProcessIfEmptyLocation`).

The last option allows processing of spreadsheets that may contain a large amount of missing values. The option indicates that the M^2 language processor should, if possible, conservatively drop the sub-expression containing the empty reference rather than dropping the entire expression. Consider, for example, the following M^2 expression declaring an individual from cell A5 of a spreadsheet and associating a property `hasAge` with it using the value in cell A6:

Individual: @A5 Facts: hasAge @A6(mm:ProcessIfEmptyLocation)

Here, using the default skip behavior action, a missing value in cell A5 will cause the expression to be skipped. However, the process directive for the `hasAge` property value in cell A6 will instead drop only the sub-expression containing it if that cell is empty. So, if cell A5 contains a value and cell A6 is empty, the resulting expression will still declare an individual.

Using a similar approach, more fine grained empty value handling is also supported to specify different empty value handling behaviors for `rdf:ID` and `rdfs:label` values. Here, the label directives are `mm:ErrorIfEmptyLabel`, `mm:SkipIfEmptyLabel`, `mm:WarningIfEmptyLabel`, and `mm:ProcessIfEmptyLabel` with equivalent keywords for RDF identifier handling.

One additional option is provided to deal with empty cell values. This option is targeted to the common case in many spreadsheets where a particular cell is supplied with a value and all empty cells below it are implied to have the same value. In this case, when these empty cells are being processed, their location must be 'shifted' to the location above it containing a value. For example, the following expression uses this keyword to indicate that call A5 does not contain a value for the name of the declared class then the row number must be shifted upwards until a value is found:

Class: @A5(mm:ShiftUp Drug)

If no value is found, normal empty value handling processing applied. Similar directives provide for shifting down (`mm:ShiftDown`), and to allow shifting to the left (`mm:ShiftLeft`) or to the right (`mm:ShiftRight`).

3.2 M^2 Mapping Process

The M^2 mapping process takes a source spreadsheet, set of M^2 expressions, and target ontology as input, and 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 populates 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 named 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 does 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.

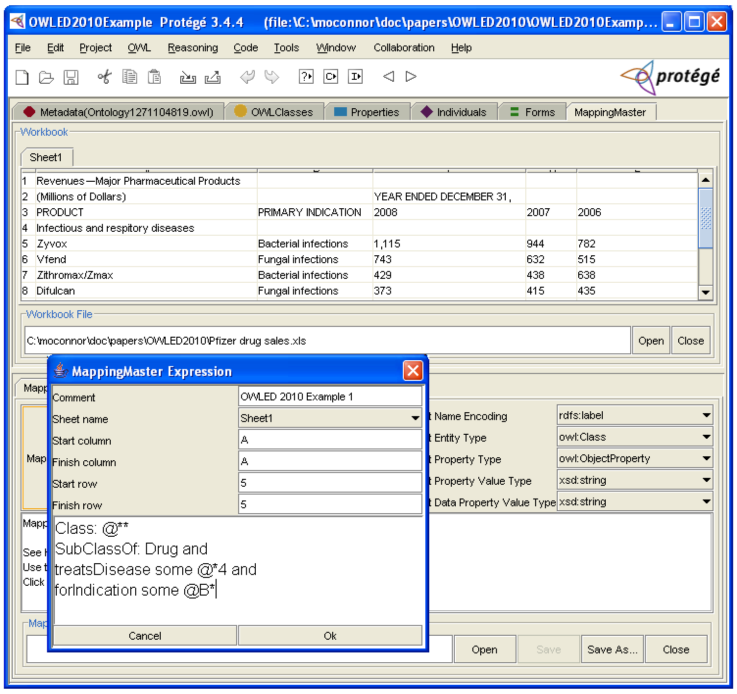
M^2 provides an *option specification clause* for each option type. The general form of this option specification clause is a keyword followed by a value. For example, the default name encoding for all mappings can be written:

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 OWL reasoner after each expression is executed.

4 Implementation

We have developed a parser, editor, and a mapper for the M^2 DSL. 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 (Figure 3).


OWLED2010Example Protégé 3.4.4 (file:\C:\moconnor\doc\papers\OWLED2010\OWLED2010Examp...
File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(Ontology1271104819.owl) OWLClasses Properties Individuals Forms MappingMaster
Workbook
Sheet1
Workbook File
Open Close
MappingMaster Expression
Cancel Ok
Open Save Save As... Close

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

C:\moconnor\doc\papers\OWLED2010\Pfizer drug sales.xls

Comment	OWLED 2010 Example 1
Sheet name	Sheet1
Start column	A
Finish column	A
Start row	5
Finish row	5

```
Class: @**
SubClassOf: Drug and
treatsDisease some @*4 and
forIndication some @B*
```

Name Encoding	rdfs:label
Entity Type	owl:Class
Property Type	owl:ObjectProperty
Property Value Type	xsd:string
Data Property Value Type	xsd:string

Figure 3. Screen shot of the Mapping Master Protégé plug-in in Protégé-OWL. The top half of the plug-in screen shows the preview screen, which allows users to explore Excel or CSV-based spread sheets. The bottom portion of the plug-in screen shows the configuration control panel, which allows users to set default options for the mapping process and to initiate the mapping and review results. The floating popup shows the M^2 expression editor.

The 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 M^2 is also provided. M^2 expressions can also be defined interactively and then executed to map the contents of loaded spreadsheets to a target ontology. The plug-in also includes a persistence mechanism to save and reload these mappings.

5 Empirical Evaluation

We evaluated M^2 on a variety of third party spreadsheets from several domains. Here, we describe the experiences encountered during these evaluations. One evaluation was performed by the authors on a range of publicly available financial spreadsheets and two were performed in collaboration with other research groups. All three required the generation of OWL ontologies, which ranged from simple ontologies containing basic class and property declarations to ontologies containing definitions of complex necessary and sufficient conditions. The source spreadsheets ranged from entity-per-row layouts to spreadsheets containing irregular non-tabular structures.

5.1 Financial Spreadsheets

As discussed in Section 1, financial analysts make extensive use of spreadsheets that often contain extremely varied data layouts. In product sales spreadsheets, for example, a tabular layout of core numerical data is common, which can be associated with surrounding data in complex ways. Using a range of publicly available spreadsheets of this type, we evaluated the ability of M^2 to map them to OWL.

The spreadsheet presented in Figure 1 is a typical example. It shows a set of sales figures contained in the grid C5:E8. Each grid cell contains the sales amount for a particular drug in a particular year. The year for each cell is contained in row 3 of its column and the name of the relevant drug in column A of its row. The heading in column A above each drug indicates its category, while column B for each cell row contains the primary use of the named drug. A financial analyst wishing to model this information in OWL must represent both the drug and sales information and associate them with each other. To generate these definitions, an OWL expression must be defined for each drug and sales amount. Ex.1 and 2 show a possible set of expressions that define a single sale and its associated drug.

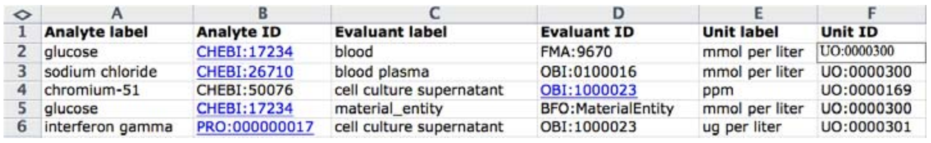
Figure 4 shows the two M^2 mapping expressions for these examples. In these expressions, the drug subclass expression is iterated from rows 5 to 8 of column A and the sales subclass expression is iterated over the grid C5:E8.

```
Class: @A* SubClassOf: Drug and
                       treatsDisease some @A4 and forIndication some @B*
Class: @** SubClassOf: SalesAmount and
                       forDrug some @A* and amount has @**
```

Figure 4. Two M^2 expressions to map content of a financial spreadsheet (Figure 1) to OWL expressions (Ex. 1 and 2). In this example, the drug subclass expression is iterated from rows 5 to 8 of column A and the sales subclass expression is iterated over the grid C5:E8.

5. 2 Ontology for Biomedical Investigations

The Ontology for Biomedical Investigations Consortium (OBI; [12]) is producing an integrated ontology for the description of life science and clinical investigations. The group has developed a spreadsheet-based procedure to allow domain experts to add terms to the OBI ontology that supports complex logical definitions yet is relatively simple to use for non ontology specialists. The procedure is based on editing definitions in a spreadsheet format, which is subsequently converted to OWL. An example spreadsheet is shown in Figure 5.



◇	A	B	C	D	E	F
1	**Analyte label**	**Analyte ID**	**Evaluant label**	**Evaluant ID**	**Unit label**	**Unit ID**
2	glucose	CHEBI:17234	blood	FMA:9670	mmol per liter	UO:0000300
3	sodium chloride	CHEBI:26710	blood plasma	OBI:0100016	mmol per liter	UO:0000300
4	chromium-51	CHEBI:50076	cell culture supernatant	OBI:1000023	ppm	UO:0000169
5	glucose	CHEBI:17234	material_entity	BFO:MaterialEntity	mmol per liter	UO:0000300
6	interferon gamma	PRO:000000017	cell culture supernatant	OBI:1000023	ug per liter	UO:0000301

Figure 5. An example OBI spreadsheet for submitting analyte assay term definition. These definitions include classes defined in several external ontologies, such as the Chemical Entities of Biological Interest (ChEBI) ontology and the Foundational Model of Anatomy (FMA).

It contains several example entities necessary to specify an *analyte assay*. An analyte assay measures the concentration of a molecular entity in a material entity, such as measuring glucose concentration in blood. Each row in the spreadsheet contains the information necessary to define a single assay with specific columns containing its parameters. Each OBI OWL definition of an assay relates the material in which the concentration is measured (the *evaluant*; e.g., blood), the molecular entity that is detected (the *analyte*; e.g., glucose), and the units of the measurement being made (e.g., microgram per liter). For example, the ‘Analyte label’ column is expected to contain the name of the analyte, and the ‘Evaluant ID’ column the name of the evaluant. The content of these cells contains the URIs of terms in external ontologies.

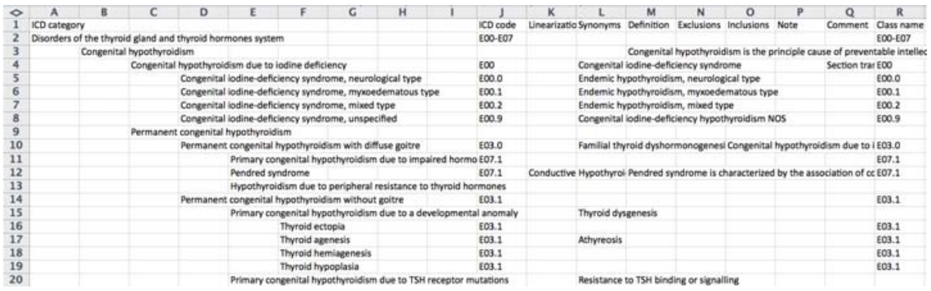
Figure 6 shows the M^2 expression that generates the analyte assay definition containing necessary and sufficient conditions from this spreadsheet. As can be seen, the contents of columns A, B, D, and F are specified by references in this expression and are incorporated into the definition of each assay. In this expression, a default `rdf:ID` name encoding is used to resolve the URIs in columns B, D, and F; the analyte assay class uses `rdfs:label` encoding for its name. This single M^2 expression replaces a custom script that was developed by an OBI developer and was written and executed using the Mapping Master Protégé plug-in. The OBI team is currently using this tool to develop a range of additional mappings.

```
Class: @A*(rdfs:label 'analyte assay')
EquivalentTo:
(achieves_planned_objective some 'analyte measurement objective') and
(realizes some ('evaluant role' and (role_of some
                                    @D*(material_entity)))) and
(realizes some ('analyte role' and
                (role_of some ('scattered molecular aggregate' and
                              ('has grain' only
                               @B*('molecular entity'))))))
SubClassOf:
 has_specified_output some
     ('scalar measurement datum' and
      ('is quality measurement of' some 'molecular concentration') and
       ('has measurement unit label' some
        @F*('measurement unit label')))
```

Figure 6. M^2 expressions to map content of OBI spreadsheet to necessary and sufficient definitions of analyte assays in the OBI ontology. Here, the expression is iterated from row 2 to the end of the spreadsheet.

5.3 International Disease Classification

The International Statistical Classification of Diseases and Related Health Problems (ICD; [16]) provides a standard classification for diseases and a wide variety of health-related indications. The standard has gone through a variety of revisions with the 11th revision (ICD-11) due in 2015. A key to the development of ICD-11 is a content model designed to support detailed descriptions of the characteristics of each disease category and clear relationships to other terminologies, all of which are to be modeled in OWL. Part of this content model is to be populated with definitions previously developed and encoded in spreadsheets and primarily describe individual disease classifications. A sample of these definitions is presented below in Figure 7.



◇	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	ICD category									ICD code	Linearizatio	Synonyms	Definition	Exclusions	Inclusions	Note	Comment	Class name
2	Disorders of the thyroid gland and thyroid hormones system									E00-E07								E00-E07
3		Congenital hypothyroidism											Congenital hypothyroidism is the principle cause of preventable intellec					
4			Congenital hypothyroidism due to iodine deficiency							E00		Congenital iodine-deficiency syndrome					Section trar	E00
5				Congenital iodine-deficiency syndrome, neurological type						E00.0		Endemic hypothyroidism, neurological type						E00.0
6				Congenital iodine-deficiency syndrome, myxoedematous type						E00.1		Endemic hypothyroidism, myxoedematous type						E00.1
7				Congenital iodine-deficiency syndrome, mixed type						E00.2		Endemic hypothyroidism, mixed type						E00.2
8				Congenital iodine-deficiency syndrome, unspecified						E00.9		Congenital iodine-deficiency hypothyroidism NOS						E00.9
9			Permanent congenital hypothyroidism															
10				Permanent congenital hypothyroidism with diffuse goitre						E03.0		Familial thyroid dyshormonogenesi			Congenital hypothyroidism due to i			E03.0
11					Primary congenital hypothyroidism due to impaired hormo					E07.1								E07.1
12					Pendred syndrome					E07.1	Conductive	Hypothyroi	Pendred syndrome is characterized by the association of cc					E07.1
13					Hypothyroidism due to peripheral resistance to thyroid hormones													
14				Permanent congenital hypothyroidism without goitre						E03.1								E03.1
15					Primary congenital hypothyroidism due to a developmental anomaly							Thyroid dysgenesis						
16						Thyroid ectopia				E03.1								E03.1
17						Thyroid agenesis				E03.1		Athyreosis						E03.1
18						Thyroid hemiagenesis				E03.1								E03.1
19						Thyroid hypoplasia				E03.1								E03.1
20					Primary congenital hypothyroidism due to TSH receptor mutations							Resistance to TSH binding or signalling						

Figure 7. Spreadsheet containing ICD disease classes, their position in a disease hierarchy, and data about each disease.

Column R contains the disease classes to be constructed. The OWL content model requires that the content of this column is used as the `rdf:ID` of the created class if a value if present for a row; if no value is present, an `rdf:ID` should be automatically generated for the new class. The class's label comes from the closest value to its left in columns A through I. These rows contain a hierarchy of the classes, with the leftmost column containing the root classes. As can be seen, most of the cells in this range are empty and should therefore be skipped. Property values for each created class are also added to the class created for column R using columns L-O (which will be OWL Full). Four object properties (`synonym`, `definition`, `exclusion`, `inclusion`) are associated with each generated class (with types `SynonymTerm`, `DefinitionTerm`, `ExclusionTerm`, `InclusionTerm`) with individuals generated for each. Each individual is assigned a property called `label` from the contents of the cell in its column. Again, if these cells contain no values for a particular row, they are skipped for that property (but other properties for that class get any values that are present).

In association with the developers of content model, we developed a collection of M^2 expressions to support mapping these spreadsheets to OWL, which replaced a custom mapping script they had previously developed. Figure 8 shows a sample of the resulting M^2 expressions. The first expression declares the disease class using the cell content as its `rdf:ID` and sets its `rdfs:label` to the closest value in columns A through I. The second expression declares the subclass pairs for columns A and B. There is one of these expressions for each adjacent column pair. The third expression associates an individual with four property values using columns L through M. The final expression sets the `label` property for one of these individuals. There is one expression for each of the four property value individuals.

```
Class: @R*(rdf:ID rdfs:label=(@I*(mm:ShiftLeft)))

Class: @B*(rdf:ID=(@R*) rdfs:label mm:SkipIfEmptyLabel)
 SubClassOf: @A*(rdf:ID=(@R*) rdfs:label mm:SkipIfEmptyLabel mm:ShiftUp)

Individual: @R*(Class rdf:ID rdfs:label=(@I*(mm:ShiftLeft)))
 Facts: synonym    @L*(mm:SkipIfEmptyLocation SynonymTerm)
        definition @M*(mm:SkipIfEmptyLocation DefinitionTerm )
        exclusion  @N*(mm:SkipIfEmptyLocation ExclusionTerm)
        inclusion  @O*(mm:SkipIfEmptyLocation InclusionTerm)

Individual: @L*(mm:SkipIfEmptyLocation SynonymTerm)
 Facts: label @L*
```

Figure 8. M^2 expressions to map the content of a spreadsheet to an ICD content model. These expressions are iterated from row 2 to the final row of the spreadsheet.

5.4 Evaluation Results

In summary, real-world use of the M^2 language and mapping process has demonstrated that it provides a compact, user-friendly mechanism for mapping complex spreadsheets to OWL. During the evaluation, we found that M^2 was expressive enough to capture the desired mappings in all examples. Further, the use cases demonstrated the utility of the language's novel features not supported in previous spreadsheet mapping work.

6 Conclusion

Recent approaches for 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. This mapping language supports arbitrary spreadsheet cell references and provides a compact, user-friendly, OWL-centric 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. As described in this paper, this plug-in has been used successfully by several research groups.

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 and by the National Library of Medicine under grants LM007885 and LM009607. We would also like to thank Peter Moore, Natasha Noy, Samson Tu, and members of the OBI Consortium. An initial, shorter version of this paper appeared in [18].

References

[1] Langegger, A., Woss, W. XLWrap-querying and integrating arbitrary spreadsheets with SPARQL. ISWC 2009, LNCS 5823, Springer, 2009.
[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. ISWC 2008, LNCS 5318, 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.mindswap.org/~mhgoeve/convert/
[8] Huynh, D., Karger, D., Miller, R. Exhibit: lightweight structured data publishing. 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
[13] Bizer, C.: D2R MAP - A Database to RDF Mapping Language. 12th International World Wide Web Conference. Budapest, Hungary, 2003.
[14] Gennari, J. Musen, M., Fergerson, R., Grosso, W., Crubezy, M. Eriksson, H. Noy, N., and Tu, S. The evolution of Protégé-2000: An environment for knowledge-based systems development. International Journal of Human-Computer Studies, 58(1):89–123, 2003.
[15] Knublauch, H. An AI tool for the real world: Knowledge modeling with Protégé. JavaWorld, June 20, 2003.
[16] World Health Organization. Production of ICD-11: The overall revision process: http://www.who.int/classifications/icd/ICDRevision.pdf, 2007.
[17] Mapping Master BNF: http://swrl.stanford.edu/MappingMaster/1.0/BNF/MappingMasterParser.html
[18] O’Connor, M.J., Halaschek-Wiener, C., Musen, M. M2: a Language for Mapping Spreadsheets to OWL OWL: Experiences and Directions (OWLED), Sixth International Workshop, San Francisco, CA, 2010.