Specification of the Schema of Spreadsheets for the Materialization of Ontologies from Integrated Data Sources

Sergio Alejandro Gómez[1,2](✉) and Pablo Rubén Fillottrani[1,2]

[1] Laboratorio de I+D en Ingeniería de Software y Sistemas de Información (LISSI),
Departamento de Ciencias e Ingeniería de la Computación,
Universidad Nacional del Sur, San Andrés 800, Bahía Blanca, Argentina
{sag,prf}@cs.uns.edu.ar

[2] Comisión de Investigaciones Científicas de la Provincia de Buenos Aires
(CIC-PBA), La Plata, Argentina
https://lissi.cs.uns.edu.ar/

Abstract. In Ontology-Based Data Access (OBDA), a knowledge base known as an ontology models both the problem domain and the underlying data sources. We are concerned with providing with tools for performing OBDA with relational and non-relational data sources. We developed an OBDA tool that is able to access H2 databases, CSV files and Excel spreadsheets allowing the user to explicitly formulate mappings, and populating an ontology that can be saved for later querying. In this paper, we present a language for specifying the schema of the data in a spreadsheet data application, which then can be used to access the contents of a set of Excel books with the ultimate goal of materializing its data as an OWL/RDF ontology. We characterize the syntax and semantics of the language, present a prototypical implementation and report on the performance tests showing that our implementation can handle a workload of Excel tables of the order of ten thousand records. We also show a case study in which the ontology of an idealized university library can be defined using the our tool integrating both relational and spreadsheet data.

Keywords: Ontology-based data access · Ontologies · Relational databases · Spreadsheets

1 Introduction

Despite their simplicity and ubiquity, spreadsheets are still relevant because they provide a semi-structured, distributed way of representing the information of an organization when there is no formal database; even, many times, in spite of the existence of a centralized system, informal or operational information not covered by the main system is managed in spreadsheets. Although the spreadsheet

P. Pesado and J. Eterovic (Eds.): CACIC 2020, CCIS 1409, pp. 247–262, 2021.
https://doi.org/10.1007/978-3-030-75836-3_17

applications (such as MS Excel, Apache Open Office, or Libre Office) give the possibility of making totalizations and filters, these tools allow limited functionality and are difficult to integrate with the rest of the organization's information, having to resort to data mining and datawarehousing solutions that are not always straightforwardly useful for the layman.

Ontology-based data access [1] is a prominent approach to accessing the content of heterogeneous and legacy databases that has gained relevance in the past years in which the database schema along with the semantics of the business model they are exposed as an OWL ontology and the data as RDF triples in distributed form on the web that can be queried through SPARQL end-points.

In this research, we are interested in studying formal models and novel ways of performing OBDA, with the goal of providing concrete implementations. In this sense, in recent times, we have been developing a prototype that allows to export the schema of a relational database in H2 format as an OWL ontology and its relational instance as an RDF graph, also allowing the expression of mappings to define concepts from of complex SQL queries [2]. In this paper, we present an extension to our OBDA prototype that allows a user to specify a spreadsheet application using a schema definition language. This language allows a naive user to specify the format of the data in the tables contained in sheets of several books, indicating the orientation of the tables, format of columns and rows, cross-relations between tables and books. This allows the spreadsheets to be interpreted as databases and ultimately being integrated with the rest of the OBDA application. We assume that the reader has a basic knowledge of Description Logics (DL) [3], relational databases and the Web Ontology Language [4].

This work consolidates and extends results presented in [5]. As extension of that work, we now include a discussion of how the GF OBDA systema can be used to integrate and query information of a university library composed in terms of relational and spreadsheet data where public open data has to be machine processed.

The rest of the paper is structured as follows. In Sect. 2, we present a framework for conceptual modeling of spreadsheets as ontologies. In Sect. 3, we show an empirical evaluation of the performance of the prototype creating tables and ontologies from several Excel files of increasing size. In Sect. 4, we describe a possible solution for the publication on the Semantic Web of data from a hypothetical university library where its data comes from several heterogeneous sources. In Sect. 5, we discuss related work. Finally, in Sect. 6, we conclude and foresee future work.

2 A Framework for Representing Spreadsheets

Now we present a theoretical framework to represent the data of a spreadsheet application. Later, with this framework, we will define a language to describe the schema of the data. Such a schema will be used to access the contents of the spreadsheets, interpret them, generate an SQL script, create and populate an H2 database such script, and then materialize an OWL/RDF ontology with the contents of such a database. This ontology could then be queried via a SPARQL processor (see Fig. 1). We provide the syntax of the data description language in the spreadsheet application using a BNF grammar and give its operational semantics in terms of this framework. We will use a running example throughout the article to illustrate how to use it.

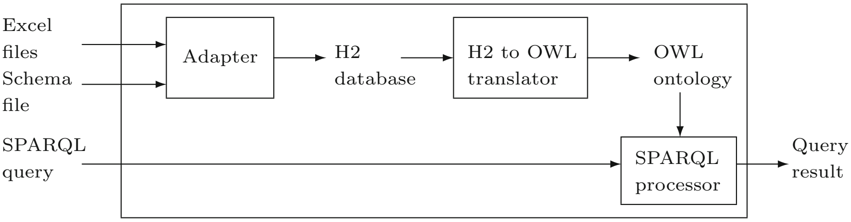

Fig. 1. Architecture of the system

A spreadsheet application data is a set of books. More formally:

Definition 1. *An spreadsheet application A is a pair* (*books*, *m*) *where books is a set of books and m is a map from a unique identifier into an object of the application.*

A book is basically a set of sheets along with further information. Formally:

Definition 2. *A* book *b is a tuple* (*id*, *path*, *sheets*, *sheetByID*) *where id is the identifier of the book, path is the absolute path of the Excel file defining the book, sheets is a list of sheets, and sheetByID is a map from sheet identifier into a sheet.*

A sheeet is composed by a set of tables. Formally:

Definition 3. *A* sheet *s is a tuple* (*id*, *name*, *tables*, *tableByID*, *container BookID*) *where id is the unique identifier of the sheet, name is the sheet's name in the container Excel book, tables is the set of tables contained in this sheet, tableByID is a map from unique table identifier into a table, and containerBookID is the identifier of the book containing the sheet.*

A table has a header, a set of records, and has an orientation (either horizontal or vertical). A cell range defines a rectangle of the data sheet specified by two cell references. Tables can contain references to other tables. Formally:

Definition 4. *A* table *t is a tuple* (*id*, *className*, *orientation*, *initialDataCell*, *finalDataCell*, *initialHeaderCell*, *finalHeaderCell*, *headerInfo*, *indexOfKeyField*, *crossReferences*, *containerSheetID*, *containerBookID*) *where id is the unique identifier of the table, className is the class in the target ontology defined by the table, orientation is either vertical or horizontal, initialDataCell is the* *top-left corner of the table's data, finalDataCell is the bottom-right corner of the table's data, initialHeaderCell is the top-left corner of the table's header, finalHeaderCell is the bottom-right corner of the table's header, headerInfo is a map from integer i into a header datum object* h_i*, crossReferences is a set of cross-references from this table into other tables, containerSheetID is the identifier of the sheet containing this table, and containerBookID is the identifier of the book containing this table. A* header datum *is a tuple* $(i, name, type)$ *where i is the 1-based index of the header datum in its container map, name is the name of the field, and type is the type of the field, that can be one of string, numeric (either integer or real), boolean, or date. A* cell *has a row (a positive number) and a column (a 1-based positive number). A* range *is pair* (c_i, c_f) *composed of an initial cell* c_i *and a final cell* c_f*. A* cross-reference *is a tuple* (i, t, j) *where i is the index of the field in the source table, t is the identifier of the destination table and j is the index of the field in the destination table.*

2.1 Grammar for the Spreadsheet Description Language

We need a language for expressing the elements of this framework. Let us consider the spreadsheet in Fig. 2 containing two tables representing people and their cell phones. We will use that example in order to introduce the elements of our language for describing the schema of the data in the spreasheet with the goal of materializing an ontology from it so it can be queried by means of SPARQL. We now define the grammar for writing scripts for defining the structure of Excel application data. We discuss each construct by giving its meaning, the BNF rules that defines its syntax and an example describing its elements.

	A	*B*	*C*	*D*	*E*	*F*	*G*
1							
2		**PersonID**	**Name**	**DateOfBirth**	**Checked**	**Weight**	**Status**
3		1	John	1/1/1981	TRUE	100.5	heavy
4		2	Mary	2/2/1982	FALSE	60.5	light
5		3	Paul	3/3/1983	TRUE	80.5	heavy
6							
7							
8		**CellID**	1	2	3	4	
9		**Brand**	Samsung	Apple	Nokia	Samsung	
10		**Model**	S8	Iphone 11	1100	J7	
11		**Owner**	1	2	1	2	

Fig. 2. A spreadsheet named *Data* representing a set of people and their cell phones

A script is sequence of commands and is the start symbol of the grammar:

$\langle \textit{script} \rangle$ ::= $\langle \textit{command} \rangle$*

There are several available commands to be used in the description of schemas of Excel files.

$\langle \textit{command} \rangle$ ::= $\langle \textit{book-declaration} \rangle$ | $\langle \textit{sheet-declaration} \rangle$ |$\langle \textit{table-declaration} \rangle$
| $\langle \textit{table-header-declaration} \rangle$ | $\langle \textit{table-data-declaration} \rangle$ | $\langle \textit{table-field-declaration} \rangle$
| $\langle \textit{table-key-field-declaration} \rangle$ | $\langle \textit{cross-ref-declaration} \rangle$ | $\langle \textit{comment} \rangle$

A book can be declared by giving it an identifier and a path. Identifiers are sorrounded by quotation marks and are composed in the usual way.

⟨*book-declaration*⟩ ::= book ⟨*id*⟩ has-path ⟨*path*⟩

⟨*id*⟩ ::= "⟨*identifier*⟩"

⟨*identifier*⟩ ::= ⟨*letter*⟩.(⟨*letter*⟩|⟨*digit*⟩)*

⟨*letter*⟩ ::= a | b | . . . |z | A | B | . . . | Z

⟨*digit*⟩ ::= 0 | 1 | . . . | 9

⟨*path*⟩ ::= ". . . windows file path . . . "

Example 1. Consider the piece of code that expresses that book b_1 has as its path the Excel file `book1.xlsx` located in the `Escritorio8` subfolder in the desktop folder: `book "b1" has-path "c:/users/sgomez/Desktop/Escritorio8/book1.xlsx"`.

A book has at least one data sheet. Each sheet has an identifier in this schema file, a name in the spreadsheet and it is located in a book.

⟨*sheet-declaration*⟩ ::= sheet ⟨*id*⟩ name ⟨*id*⟩ in ⟨*id*⟩

Example 2. Consider the code: `sheet "s1" name "Data" in "b1"`. It expresses that the spreadsheet s_1 has been named *Data* and it is located in the book b_1.

Each spreadsheet can have several tables. Each table has an identifier, is contained in a certain spreadsheet, defines a class and has an orientation which either is horizontal or vertical.

⟨*table-declaration*⟩ ::= table ⟨*id*⟩ in-sheet ⟨*id*⟩ class-name ⟨*id*⟩ orientation ⟨*orientation-literal*⟩

⟨*class-name*⟩ ::= ⟨*id*⟩

⟨*orientation-literal*⟩ ::= horizontal | vertical

Example 3. Consider the commands: `table "t1" in-sheet "s1" class-name "Person" orientation vertical` and `table "t2" in-sheet "s1" class-name "Phone" orientation horizontal`. They define that there are two tables: t_1 and t_2, which are both located in sheet s_1. Table t_1 defines a class name *Person* while table t_2 defines a class named *Phone*. The orientation of t_1 is vertical but the orientation of t_2 is horizontal.

Every table defition is composed of header and data sections, with syntax:—

⟨*table-header-declaration*⟩ ::= header ⟨*id*⟩ range ⟨*range-specification*⟩

⟨*table-data-declaration*⟩ ::= data ⟨*id*⟩ range ⟨*range-specification*⟩

⟨*range-specification*⟩ ::= "⟨*cell-spec*⟩:⟨*cell-spec*⟩"

⟨*cell-spec*⟩ ::= $\langle \textit{letter} \rangle^{+} \langle \textit{digit} \rangle^{+}$

Example 4. Consider the commands for defining the limits of tables t_1 and t_2: `header "t1" range "b2:g2"`, `data "t1" range "b3:g5"`, `header "t2" range "b8:b11"`, and `data "t2" range "c8:f11"`.

Fields are declared specifying the table to which they belong, an index, a name and a type. There is an special field called the key field:

⟨*table-field-declaration*⟩ ::= field ⟨*id*⟩ index ⟨*positive-integer*⟩ name ⟨*id*⟩ type ⟨*type-id*⟩

⟨*type-id*⟩ ::= integer | string | date | real

⟨*table-key-field-declaration*⟩ ::= key-field ⟨*id*⟩ index ⟨*positive-integer*⟩

⟨*positive-integer*⟩ ::= (1..9)⟨*digit*⟩*

Example 5. Consider the piece of code for defining the fields of tables t_1 and t_2:

```
field "t1" index "1" name "PersonID" type integer
field "t1" index "2" name "Name" type string
field "t1" index "3" name "DateOfBirth" type date
field "t1" index "4" name "Checked" type boolean
field "t1" index "5" name "Weight" type real
field "t1" index "6" name "Status" type string
key-field "t1" index "1"
field "t2" index "1" name "CellID" type integer
field "t2" index "2" name "Brand" type string
field "t2" index "3" name "Model" type string
field "t2" index "4" name "Owner" type integer
key-field "t2" index "1"
```

The table t_1 has 6 fields named *PersonID*, *Name*, *DateOfBirth*, *Checked*, *Weight* and *Status* of type integer, date, boolean, real and string, resp. The table t_2 has 4 fields named *CellID* and *Owner* both of type integer, and *Brand* and *Model* of type string. The key field of t_1 is *PersonID* while the key field of t_2 is *CellID*. Notice that no indications are given here if the contents of a cell is either a formula or a value and it is neither necessary. For instance the column *Status* is a formula of the form: `=IF(F3>=80, "heavy", "light")` indicating that if the weight of the person is greater than or equal to 80 kg, the person is considered as heavy, otherwise is deemed as light.

A table can have cross-references to other tables.

⟨*cross-ref-declaration*⟩ ::= cross-ref from ⟨*id*⟩ index ⟨*positive-integer*⟩ into ⟨*id*⟩ index ⟨*positive-integer*⟩

Example 6. The following piece of code defines a cross-reference from field number 4 of table t_2 into field number 1 of table t_1:

```
cross-ref from "t2" index "4" into "t1" index "1"
```

One-line comments are allowed in our scripting language and they begin with the hashtag character.

⟨*comment*⟩ ::= #⟨*character*⟩*

⟨*character*⟩ ::= any Ascii character excluding end of line

2.2 Semantics of Spreadsheet Constructors

The semantics of the empty spreadsheet application *create* is $(\{\},\{\})$. The semantics of commands is given in terms of the function Sem from commands by spreadsheet applications into spreadsheet applications. The semantics of a book declaration is as follows:

$$\mathsf{Sem}(\text{sheet "}id\text{" name "}n\text{" in "}bid\text{"}, (books, m)) = (books', \{(id, s)\} \cup m) \text{ where}$$

$$\begin{aligned} books' &= books - \{b\} \cup \{b'\} \\ b &= m(bid) = (bid, p, sheets, sheetByID), \\ b' &= (bid, p, \{s\} \cup sheets, \{(id, s)\} \cup sheetByID) \\ s &= (id, n, \{\}, \{\}, bid) \end{aligned}$$

The semantics of the declaration of a table *id*, in sheet *sid*, determining a class c, with orientation o, with n fields named $name_1$, ..., $name_n$ of types t_1, ..., t_n, key field k, m cross-references from fields i_1, ldots, i_m into foreign tables tid_1, ..., tid_m and foreign fields with indexes j_1, ..., j_m, resp., header info in the range $h_1 : h_2$ and data info in the range $d_1 : d_2$ is given shown in Fig. 3.

$$\mathsf{Sem}(sec, (books, m)) = (books', \{(id, t)\} \cup m) \text{ where}$$

$$\begin{aligned} sec &= (\text{table "}id\text{" in-sheet "}sid\text{" class-name "}c\text{" orientation } o\boxed{\hookleftarrow} \\ &= \text{header "}id\text{" range "}h_1 : h_2\text{"}\boxed{\hookleftarrow} \\ &\quad \text{field "}id\text{" index "1" name "}name_1\text{" type } t_1\boxed{\hookleftarrow} \\ &\quad \ldots \\ &\quad \text{field "}id\text{" index "}n\text{" name "}name_n\text{" type } t_n\boxed{\hookleftarrow} \\ &\quad \text{key-field "}id\text{" index "}k\text{"}\boxed{\hookleftarrow} \\ &\quad \text{data "}id\text{" range "}d_1 : d_2\text{"}\boxed{\hookleftarrow} \\ &\quad \text{cross-ref from "}id\text{" index "}i_1\text{" into "}tid_1\text{" index "}j_1\text{"}\boxed{\hookleftarrow} \\ &\quad \ldots \\ &\quad \text{cross-ref from "}id\text{" index "}i_m\text{" into "}tid_m\text{" index "}j_m\text{")} \\ s &= m(id) = (sid, name, ts, tableByID, containerBookID) \\ t &= (id, c, o, d_1, d_2, h_1, h_2, head, k, cross, sid) \\ s' &= (sid, name, \{t\} \cup ts, \{(id, t)\} \cup tableByID, containerBookID) \\ books' &= books - \{b\} \cup \{b'\} \\ b &= (bid, p, sheets, sheetByID) = m(containerBookID) \\ b' &= (bid, p, sheets', sheetByID) \\ sheets' &= \{s\} \cup sheets \\ cross &= \{(i_1, tid_1, j_1), \ldots, (i_m, tid_m, j_m)\} \\ head &= \lambda i.(i, name, t_i), \text{ with } i = 1, \ldots, n \end{aligned}$$

Fig. 3. Semantics of table declaration commands

2.3 Generation of Databases and Ontologies from Spreadsheets

We now discuss the generation of OWL/RDF ontologies from spreadsheet applications. Given a book with mapping m of identifiers into objects, let t be a table such that $t = (id, c, o, d_1, d_2, h_1, h_2, head, k, cross, s)$, such that $cross = \{(i_1, tid_1, j_1), \ldots, (i_m, tid_m, j_m)\}$, and $head = \lambda i.(i, name_i, t_i)$, with $i = 1, \ldots, n$.

The SQL code in Fig. 4 represents the schema of table t, where second and sixth are the projectors of the second and the sixth components of a tuple, resp. Then this SQL code is used to materialize an H2 database, which in turn is used to materialize an OWL/RDF ontology using the methodology described in our previous work [6].

create table "c"(

"$name_1$" t_1, ..., "$name_k$" t_k primary key, ..., "$name_n$" t_n,

foreign key ("$second(head(i_1))$") references "$second(m(tid_1))$"("$second(sixth(m(tid_1))(j_1))$"),

..., foreign key ("$second(head(i_m))$") references "$second(m(tid_m))$"("$second(sixth(m(tid_m))(j_m))$"));

Fig. 4. SQL script for creating a generic table t

Example 7. The spreadsheet in Fig. 2 is represented by the SQL script in Fig. 5. Then, from this script, a database is created and the ontology materialized from that database has the following DL axioms (that are ultimately serialized as OWL/RDF): Person $\sqsubseteq \exists$PersonID, $\exists$PersonID$^-$ $\sqsubseteq$ Integer, Person $\sqsubseteq \exists$name, $\exists$name$^-$ $\sqsubseteq$ String, Person $\sqsubseteq \exists$dateOfBirth, $\exists$dateOfBirth$^-$ $\sqsubseteq$ Date, Person $\sqsubseteq \exists$checked, $\exists$checked$^-$ $\sqsubseteq$ Boolean, Person $\sqsubseteq \exists$weight, $\exists$weight$^-$ $\sqsubseteq$ Real, Person $\sqsubseteq \exists$status, $\exists$status$^-$ $\sqsubseteq$ String, Phone $\sqsubseteq \exists$cellID, $\exists$cellID$^-$ $\sqsubseteq$ Integer, Phone $\sqsubseteq \exists$brand, $\exists$brand$^-$ $\sqsubseteq$ String, Phone $\sqsubseteq \exists$model, $\exists$model$^-$ $\sqsubseteq$ String, Phone $\sqsubseteq \exists$owner $\exists$owner$^-$ $\sqsubseteq$ Integer, Phone $\sqsubseteq \exists$`ref`_owner $\exists$`ref`_owner$^-$ $\sqsubseteq$ Person. The assertions for representing the first record of the class Person are: PersonID(Person#1, 1), name(Person#1, JOHN), dateOfBirth(Person#1, 1981-01-01), checked(Person#1, TRUE), weight(Person#1, 100.5), and status(Person#1, HEAVY).

```
create table "Person"(
    "PersonID" int primary key,     "Name" varchar(50), "DateOfBirth" date,
    "Checked" boolean, "Weight" real, "Status" varchar(50) );
create table "Phone"(
    "CellID" int primary key, "Brand" varchar(50), "Model" varchar(50), "Owner" int,
    foreign key ("Owner") references "Person"("PersonID") );
insert into "Person"("PersonID", "Name", "DateOfBirth", "Checked", "Weight", "Status")
     values (1, 'John', '1981-01-01', true, 100.5, 'heavy');
insert into "Person"("PersonID", "Name", "DateOfBirth", "Checked", "Weight", "Status")
     values (2, 'Mary', '1982-02-02', false, 60.5, 'light');
insert into "Person"("PersonID", "Name", "DateOfBirth", "Checked", "Weight", "Status")
     values (3, 'Paul', '1983-03-03', true, 80.5, 'heavy');
insert into "Phone"("CellID", "Brand", "Model", "Owner") values (1, 'Samsung', 'S8', 1);
insert into "Phone"("CellID", "Brand", "Model", "Owner") values (2, 'Apple', 'Iphone 11', 2);
insert into "Phone"("CellID", "Brand", "Model", "Owner") values (3, 'Nokia', '1100', 1);
insert into "Phone"("CellID", "Brand", "Model", "Owner") values (4, 'Samsung', 'J7', 2);
```

Fig. 5. SQL code obtained from the spreadsheet in Fig. 2

3 Experimental Evaluation

We now discuss some of the tests we have performed in order to test how our application handles increasing demands in database size. The performance of our system is affected mainly by the fact that we tables are metarialized as RDF triples and also by four factors: (i) the system is implemented in the JAVA programming language; (ii) the database management system that we use is H21, (iii) the handling of the global ontology is done via the OWL API [7], and (iv) the access to the Excel files is implemented using the Apache POI library [8]. Our tests were conducted on an ASUS notebook having an Intel Core i7, 3.5 GHz CPU, 8 GB RAM, 1 TB HDD, and Windows 10. They involved the creation of databases with single table extracted from Excel books containing only a sheet with a table containing 100 fields of numeric type filled with an increasing number of records. In Table 1, we can see the times for loading the Excel files and the size of the materialized ontologies. Therefore, we conclude that our application can only handle tables with a size of tens of thousands records and is not able of handling tables of a hundred thousand records.

Table 1. Running times for ontology generation from Excel files

Number of records	Excel file size [Megabytes]	Time for loading Excel file [seconds]	Time for creating ontology [seconds]	Size of ontology file [Megabytes]
10	0.012	0.901	0.276	0.115
100	0.033	1.774	0.359	0.910
1,000	0.255	5.825	1.067	8.951
10,000	2.640	29.703	4.253	90.951
100,000	26.742	Out of memory error		

4 Case Study: OBDA for Library Management

We contend that the approach for describing the schema of spreadsheet data described above can be used as the basis for the development of real-world OBDA applications allowing the publication of organization data as rich OWL/RDF ontologies. We try to validate our thesis by describing a possible solution for the publication on the Semantic Web of data from a hypothetical university library where its data comes from several heterogeneous sources.

University libraries often work with proprietary software, or spreadsheets, to represent their inventory. When the bibliographic inventory data are represented in a relational database, they can adopt ad-hoc codings representing domain peculiarities that are often difficult to extrapolate to other systems. The attention is often personal, which makes it impossible to search the literature for the material present in them as noted in [9–13].

The OBDA system called GF [2] that allows OWL/RDF ontologies to be materialized from data represented in the form of a relational database, CSV data sheet and now Microsoft Excel spreadsheet. We will use the system to establish mappings to retrieve subsets of the data from the database and to establish rich relationships between such data in the form of classes, subclasses and properties in an ontology. This ontology, together with other similar ones from other libraries, can be published on the internet and can be consulted through a SPARQL endpoint in an integrated way to search for the availability of bibliographic material, as well as on the status of their users. Thus the objective of this section is to show how an ontology like the one presented in Fig. 6 can be constructed from heterogeneous data sources containing ad-hoc encodings and then show how GF can deal with a combination of data specified as relational data and spreadsheet data.

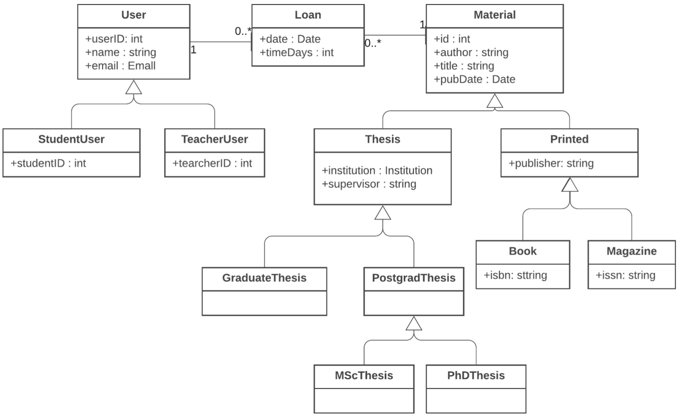

Fig. 6. Ontology for the university library

Suppose that the data of the bibliographic material, users and loans of a university library are stored in a relational database with the schema and instance as shown in Fig. 7. We see that the table that models the library loans reifies a many-to-many relationship between user and bibliographic material, which in turn is separated into 2 tables, namely, thesis and printed matter. Printed material is separated into books and magazines. The type of theses must encode variants such as graduate thesis, master's thesis and doctoral thesis. This type of simplification may, for example, need to use special values for ad-hoc encondings. For example, notice the D for codifying doctoral thesis and the M for Master Thesis, which, when querying the data using SQL, to search for doctoral thesis, requires resorting to low-level constructions such as: *select * from "Thesis" where type = "D"*.

We then will show how the use of OBDA technologies makes it possible to more naturally model the type of each document by referring to the classes and subclasses belonging to the semantics of the application domain. Ultimately all of the presented techniques can be implemented by a naive user in the GF framework of which a previous version was presented in [2] and references there in. To do this, suppose that the tables defined above are populated as in Fig. 7.

$$User(\underline{userNo}, name, email, type)$$
$$Thesis(\underline{id}, author, title, pubDate, type, institution, supervisor)$$
$$Loan(\underline{userNo}, \underline{id}, date, timeDays)$$

User

userNo	*name*	*email*	*type*
1	John	john@nosite.com	S
2	Peter	peter@nosite.com	T

Loan

userNo	*id*	*date*	*timeDays*
1	1	2020-09-01	40

Thesis

id	*author*	*title*	*pubDate*	*type*	*institution*	*supervisor*
1	Marie Curie	Recherches sur les substances radioactives	1903-01-01	D	Faculte des Sciences de Paris	Gabriel Lippmann
2	Claude Shannon	A Symbolic Analysis of Relay and Switching Circuits	1937-01-01	M	Massachusetts Institute of Technology	Vannevar Bush

Fig. 7. Relational instance of the library's database concerning Users, Theses and Loans

Consequently, when proposing a richer modeling of the domain, we are interested in defining two subconcepts of the Thesis concept called undergraduate thesis and postgraduate thesis. In turn, the postgraduate thesis concept will have two sub-concepts called MSc Thesis and PhD Thesis. Formally, we are interested in establishing the axioms in the ontology shown in Fig. 8.

$$\begin{aligned} UndergraduateThesis \sqcup GraduateThesis &\sqsubseteq Thesis \\ UndergraduateThesis \sqcap GraduateThesis &\sqsubseteq \bot \\ MScThesis \sqcup PhDThesis &\sqsubseteq GraduateThesis \\ MScThesis \sqcap PhDThesis &\sqsubseteq \bot \end{aligned}$$

Fig. 8. Axioms for classifying theses

Then, it is necessary to establish the link between the data in the tables and the concepts and assertions of the ontology. This is achieved using mappings, which are SQL expressions that define the values of the ontology assertions in terms of the values of the relational instance. In the case of theses, the mappings are as shown in Fig. 9. The definition of such mappings can be done visually in the GF frame. The system allows you to define the name of the sub-concept, from which table the data is obtained, automatically computes the SQL filter and shows the records that fill the concept. It also allows you to automatically add the axioms as shown in Fig. 8.

$$
\begin{aligned}
PhDThesis(id) &\leftarrow select\ "id\ from\ "Thesis"\ where\ "type" = 'D' \\
MScThesis(id) &\leftarrow select\ "id"\ from\ "Thesis"\ where\ "type" = 'M' \\
UndergraduateThesis(id) &\leftarrow select\ "id"\ from\ "Thesis"\ where\ "type" = 'T'
\end{aligned}
$$

Fig. 9. Mappings for defining assertions from the table *Thesis*

Suppose we have the library magazines represented in a spreadsheet like the one shown in Fig. 10. In this case, we see that the magazines table has been represented horizontally instead of vertically as it is usuallly done as GF supports both representations.

	A	*B*	*C*	*D*
...				
3		**Id**	100	101
4		**Author**	Thomas G. Rokicki	James Willis
5		**Title**	An Algorithm for Compressing Space and Time	Build Your Own Turing Machine
6		**PubDate**	4/1/2006	4/1/1981
7		**Issn**	1044-789X	0360-5280
8		**Publisher**	UBM Technology Group	UBM Technology Group
9		**Magazine**	Dr Dobb's Journal	BYTE Magazine
10		**Editor**	Andrew Binstock	Wayne Green
11				

Fig. 10. Spreadsheet called *Magazine* for representing magazines

As shown in Sect. 2.1, it is necessary to define the schema of the data prior to its import into the OBDA system. In Fig. 11, we show the schema of the spreadsheet shown in Fig. 10. As implied by Fig. 1, the system generates a relational table, which is used to generate the OWL code to update the ontology. Additionally, the axiom $Magazine \sqsubseteq Printed$ indicating that a magazine is a type of printed matter must be included. In Fig. 12, we show, as an example, the definition of the *Editorial* property of the *Magazine* class. From this spreadsheet several DL assertions are produced such as: $Magazine(101)$, $author(101, \text{James Willis})$, $editor(101, \text{Wayne Green})$, ... In Fig. 13, we show the OWL serialization of the magazine 101.

To query the data integrated in the ontology, it is necessary to use the SPARQL language [14]. For example, to find data about copies of BYTE magazine in the library, a query like the one shown in Fig. 14 can be used.

5 Related Work

XLWrap [15] constitutes an approach for generating RDF graphs of arbitrary complexity from various spreadsheet layouts, including cross tables and tables

```
book ”b1” has-path ”c:/users/john/Desktop/magazines.xlsx”
sheet ”s1” name ”Magazine” in ”b1”
table ”t1” in-sheet ”s1” class-name ”Magazine”
orientation horizontal
header ”t1” range ”b3:b10”
data ”t1” range ”c3:d10”
key-field ”t1” index ”1”
field ”t1” index ”1” name ”id” type integer
field ”t1” index ”2” name ”Author” type string
field ”t1” index ”3” name ”Title” type string
field ”t1” index ”4” name ”PubDate” type date
field ”t1” index ”5” name ”issn” type string
field ”t1” index ”6” name ”Publisher” type string
field ”t1” index ”7” name ”Magazine” type string
field ”t1” index ”8” name ”Editor” type string
```

Fig. 11. Data definition scheme for the Magazines spreadsheet

```
<owl:DatatypeProperty rdf:about=”http://foo.org/Magazine#Publisher”>
<rdfs:domain rdf:resource=”http://foo.org#Magazine”/>
<rdfs:range rdf:resource=”http://www.w3.org/2001/XMLSchema#string”/>
</owl:DatatypeProperty>
```

Fig. 12. Part of the OWL code for publishing magazines describing the Publisher property

```
<owl:NamedIndividual rdf:about=”http://foo.org/Magazine/id=101”>
<rdf:type rdf:resource=”http://foo.org#Magazine”/>
<Magazine:Author rdf:datatype=”http://www.w3.org/2001/XMLSchema#string”>
James Willis</Magazine:Author>
<Magazine:Editor rdf:datatype=”http://www.w3.org/2001/XMLSchema#string”>
Wayne Green</Magazine:Editor>
</owl:NamedIndividual>
```

Fig. 13. Part of the OWL code for 101 magazine

```
prefix foo: <http://foo.org/>
prefix r: <http://foo.org/Magazine#>

select ?r ?author ?title ?publisher
where
{
        ?r r:Magazine ”BYTE Magazine” .
        ?r r:Author ?author .
        ?r r:Title ?title.
        ?r r:Publisher ?publisher.
}
```

Fig. 14. SPARQL query to retrieve a BYTE journal.

where data is not aligned in rows. They provide a functionality similar to ours but relying in JSON for the description of data. Our approach features a simpler language geared towards naive users. *NOR2O* [16] can convert excel to Scovo and Data Cube Vocabulary but it is no longer maintained. *Excel2rdf* [1] is a Java-based command-line utility that converts Excel files into valid RDF files but as far as we know it is not possible to make precise definitions of the data contained nor export terminologies as done in our proposal. *RDBToOnto*[2] allows to automatically generate fine-tuned OWL ontologies from relational databases. A major feature of this full-fledged tool is the ability to produce structured ontologies with deeper hierarchies by exploiting both the database schema and the stored data. RDBToOnto can be exploited to produce RDF Linked Data. It can also be used to generate highly accurate RDB-to-RDF mapping rules (for D2RQ Server and Triplify). *Spread2RDF*[3] is a converter for complex spreadsheets to RDF and a Ruby-internal DSL for specifying the mapping rules for this conversion. Other solutions to the problem of wrapping Excel files into semantic technologies have migrated from the academic world to the commercial world. For example, *Open Anzo*[4] used to include both an open source enterprise-featured RDF quad store and a sophisticated service oriented, semantic middleware platform that provides support for multiple users, distributed clients, offline work, real-time notification, named-graph modularization, versioning, access controls, and transactions, giving support to applications based on W3C semantic technology standards like OWL, RDF and SPARQL. This project is no longer available it has turned into a company named *Cambridge Semantics*[5]. *TopBraid Composer*[6] can convert Excel spreadsheets into instances of an RDF schema. *TabLinker*[7] can convert non-standard Excel spreadsheets to the Data Cube vocabulary. Our work converts the contents of the records in Excel sheets to RDF but also allows to precisely define the schema of the data in OWL.

6 Conclusions and Future Work

We have presented a framework for the modeling of the schema and data of spreadsheet files by means of a description language. We have given a formal specification of the syntax of such a language with a BNF grammar and its formal semantics in terms of the framework of representation. We have shown an example of how it is used in order to explain its main components. We have also provided a prototypical implementation, showing how it is integrated into an ontology-based data access system with the aim of publishing such spreadsheets as freely available ontologies on the Semantic Web. We believe that this

[1] https://github.com/waqarini/excel2rdf.
[2] https://sourceforge.net/projects/rdbtoonto/.
[3] https://github.com/marcelotto/spread2rdf.
[4] https://www.w3.org/2001/sw/wiki/OpenAnzo.
[5] http://www.cambridgesemantics.com.
[6] https://www.topquadrant.com/knowledge-assets/faq/tbc/.
[7] https://github.com/Data2Semantics/TabLinker/wiki.

language provides a valid alternative to more technical options like JSON from which naive users can benefit while providing more control than WYSIWYG-type applications that provide similar functionality. Also, we have carried out experimental tests to determine what is the workload that our implementation can effectively handle, showing that it is viable for spreadsheets containing tables with thousands of records. We have presented a case study that shows that the approach presented in this paper can be used to integrate several data sources in heterogeneous formats to comprise a suitable alternative for the publication of data of an idealized university library.

As part of future work, we are interested in continuing to explore other types of NoSQL database models and thinking about integrating them into our ontology-based data access prototype with the aim of developing novel algorithms and techniques such as virtualization by query-rewriting to provide more flexibility in regards to volatile data than the one offered by the materialization approach.

Acknowledgments. This research is funded by Secretaría General de Ciencia y Técnica, Universidad Nacional del Sur, Argentina and by Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC-PBA).

References

1. Xiao, G., et al.: Ontology-based data access - a survey. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), pp. 5511–5519 (2018)
2. Gómez, S.A., Fillottrani, P.R.: Materialization of OWL ontologies from relational databases - a practical approach. In: Pesado, P., Arroyo, M. (eds.) Computer Science - CACIC 2019 selected papers, pp. 285–301. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-48325-8_19
3. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge (2017)
4. Bao, J., Kendall, E.F., McGuinness, D.L., Patel-Schneider, P.F.: OWL 2 Web Ontology Language Quick Reference Guide (Second Edition) W3C Recommendation, 11 December 2012 (2012)
5. Gómez, S.A., Fillottrani, P.: A language for the specification of the schema of spreadsheets for the materialization of ontologies. In Mon, A., et al. (eds.) XXVI Congreso Argentino de Ciencias de la Computación (CACIC 2020), pp. 546–555, October 2020
6. Gómez, S.A., Fillottrani, P.R.: Towards a framework for ontology-based data access: materialization of OWL ontologies from relational databases. In Pesado, P., Aciti, C., (eds.) X Workshop en Innovación en Sistemas de Software (WISS 2018), XXIV Congreso Argentino de Ciencias de la Computación CACIC 2018, pp. 857–866 (2018)
7. Matentzoglu, N., Palmisano, I.: An Introduction to the OWL API. Technical report, The University of Manchester (2016)
8. Minh, N.H.: How to Read Excel Files in Java using Apache POI (2019)
9. Dilroshan, T.C.: Identification of problems faced by university libraries in the process of automation: with special reference to the libraries of moratuwa and colombo universities. Sri Lanka J. Librarianship Inf. Manage. **1**(2), 82–98 (2009)

10. Malhan, I.: Challenges and problems of library and information education in India: an emerging knowledge society and the developing nations of Asia. Libr. Philos. Pract. **670** (2011). https://digitalcommons.unl.edu/libphilprac/670/
11. Mishra, A., Thakur, S., Singh, T.: Library automation: issues, challenges and remedies author. Times Int. J. Res. (Issue January 2015), 9–16 (2015). https://www.academia.edu/12808629/LIBRARY_AUTOMATION_ISSUES_CHALLENGES_AND_REMEDIES
12. Pothumani, S., Sridhar, J.: Solving problems of library management system. Int. J. Innov. Res. Comput. Commun. Eng. **3**(7), 6466–6469 (2015). https://doi.org/10.15680/ijircce.2015.0307167
13. Raval, A.: Problems of library automation. Int. J. Res. Educ. **2**(2) (2013). http://www.raijmr.com/ijre/wp-content/uploads/2017/11/IJRE_2013_vol02_issue_02_01.pdf
14. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language for RDF W3C recommendation, 21 March 2013 (2013). https://www.w3.org/TR/rdf-sparql-query/
15. Langegger, A., Wöß, W.: XLWrap – querying and integrating arbitrary spreadsheets with SPARQL. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 359–374. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04930-9_23
16. Terrazas, B.V., Gomez-Perez, A., Calbimonte, J.P.: NOR2O: a library for transforming non-ontological resources to ontologies. In: ESWC 2010 (2010)