A Knowledge Graph Data Transformation Model for e-Government

Friday Orji*
Department of Computer Science,
Rivers State University, Port Harcourt, Nigeria
friday.orji@ust.edu.ng

Nuka Nwiabu
Department of Computer Science,
Rivers State University, Port Harcourt, Nigeria
nwiabu.nuka@ust.edu.ng

Okoni Bennett
Department of Computer Science,
Rivers State University, Port Harcourt, Nigeria
bennett.okoni@ust.edu.ng

Onate Taylor
Department of Computer Science,
Rivers State University, Port Harcourt, Nigeria
taylor.onate@ust.edu.ng

DOI: 10.56201/ijcsmt.v10.no1.2024.pg124.142

Abstract

In the digital age, governments globally are leveraging technology to bolster their services and uplift citizen well-being. This surge has propelled the evolution of the e-Government realm, accompanied by heightened complexity, rendering it an ideal arena for exploring the widespread impact of Artificial Intelligence (AI) at large, and Knowledge Graph (KG) specifically. e-Government and AI now serve as strategic and tactical tools for governments worldwide, aiming to deliver public services with heightened efficiency, efficacy, and transparency. One area in e-Government that KG is used to address, is the challenge of creating a single knowledge source, in RDF graph data, from traditional data sources such as relational data. In this paper, we present a model for transforming data from relational to RDF, in an e-Government context. Our aim is to advance the e-Government objective of effective and efficient public service delivery and citizens engagement, given a complex e-Government instance. We focus on data transformation using RDB2RDF mapping language, a yml parser that converts mapping rules from yml to ttl turtle format. This output mapping rules is then used to transform the relational data to RDF data. Our research approach affords us the means to analyze and design our model; and validate and evaluate our work. Our model and the development approach help to achieve the e-Government goals of efficient and effective service delivery and citizens engagement.

Keywords: *Knowledge Graph, e-Government, RDF, Artificial Intelligence, Relational Data*

1. Introduction

In today's digital age, governments worldwide are harnessing technology to elevate their services and enrich the lives of their populace. This surge has propelled the emergence of the e-Government sphere, accompanied by a growing intricacy, making it an ideal arena to explore the pervasive impact of Artificial Intelligence (AI) and Knowledge Graph (KG). e-Government and AI now stand as pivotal tools, strategically and operationally, enabling governments globally to furnish public services with heightened efficiency, efficacy, and transparency [1], [2], [3], [4].

e-Government refers to the integration of Information and Communication Technology (ICT) within public policy, operations in public organizations, citizens engagement, and government services [5]. Notably, the e-Government domain is unique in the sense that it is large, heterogenous, dynamic, and shared, with different semantic world view [1]. Consequently, it serves as an ideal testing ground for cutting-edge AI innovations such as Knowledge Graphs (KG). A KG is as graphical representations of real-world entities, encompassing objects, events, situations, or concepts, KGs delineate the intricate relationships among these entities [6]. A primary way of representing KG is an organization is by using the Resource Description Framework (RDF). RDF is a primary language for developing ontologies.

Given the unique nature of e-Government, various governments have endeavored to address the challenge of data silos and divergent perspectives by adopting a One-Stop-Shop approach to government services [7]. While well-documented, these efforts to deliver seamless, integrated government services via a centralized approach have encountered limitations. Traditional One-Stop-Shop models rely on governments maintaining a singular data repository, imposing constraints on operational flexibility. Departments are compelled to utilize data from other sectors based on disparate data models, effectively operating as a single unit, which proves impractical and burdensome. The reality is that governmental departments function autonomously within distinct operational, legal, and regulatory frameworks. Without a shared semantic framework, departments cannot efficiently exchange or interpret data [8]. Additionally, the relational data model, prevalent in traditional One-Stop-Shop systems, presents inherent limitations. Complexities within relational databases hinder meaningful interpretation of data, as they lack the contextual knowledge inherent in real-world scenarios. Furthermore, the relational model often mirrors a siloed approach to real-world concepts and relationships [9].

Data and knowledge are represented in different formats, in an e-Government instance and indeed many organizations, such as relational database, Web pages, and documents. Since an organization's KG is based on RDF knowledge representation data model, a data conversion from any of the current representation into the RDF knowledge representation format, is needed. This conversion is a data lifting process [6] process which transforms data from the data level to a machine-readable knowledge level.

2. Related Work

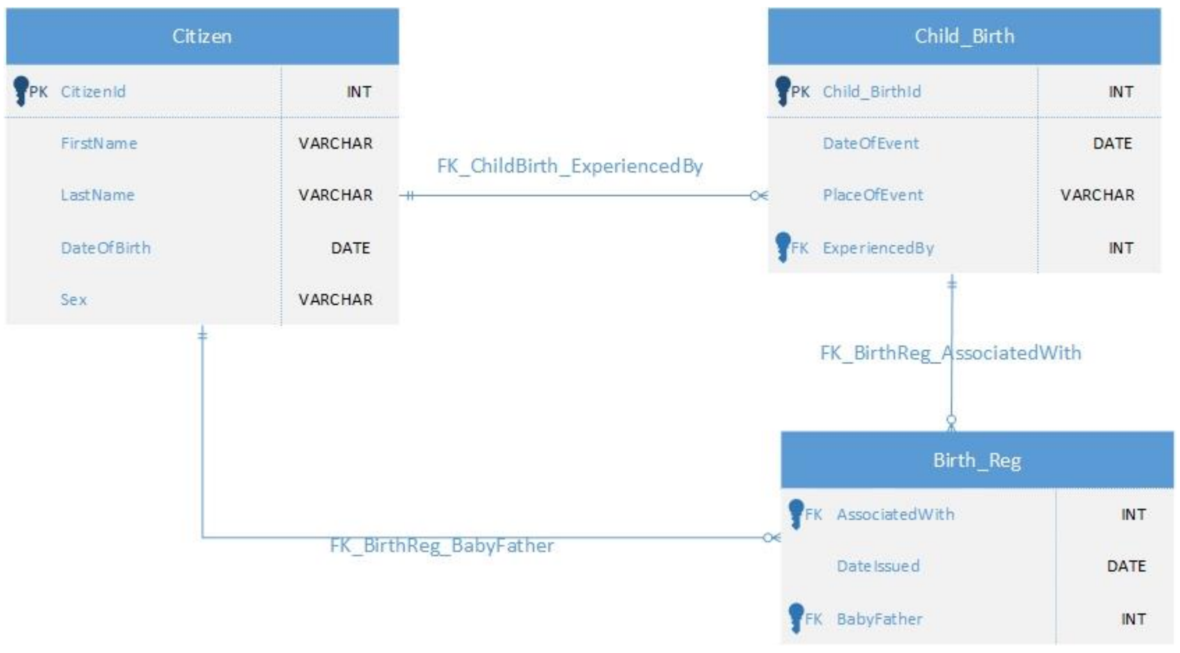
Various data lifting standard exist for different source data including approaches such as Information Extraction (IE) from natural language text [10] and the W3C standards [11] for relational data and XML data – RDB2RDF and GRDDL [12] respectively. We consider the data transformation from structured data, which is the most common form of data in most organizations' critical data infrastructure.

The W3C RDB2RDF initiative has two standards for mapping relational data into RDF. One standard is a recommendation for direct mapping of relational data to RDF [13]. This is the preferred standard, when a quick conversion is needed and the relational data schema is designed to be good enough, with well-defined primary and foreign keys, meaningful table and column names and other properties that define a good-enough relational design. The only input required, in this case, is the relational data (dataset and schema) and the output is the RDF version of the data. This conversion process is simple and straightforward, but affords little control over the conversion settings. The second standard is R2RML – RDB to RDF Mapping Language [11]. R2RML affords more control to customized the mappings to generate the RDF data based on a design. An example customization is when it is needed to generate RDF data by reusing some popular vocabularies or preferred domain ontologies.

3. Data Transformation Approach

3.1 Direct Mapping

We perform the data transformation process on the generated synthetic data (dataset and schema) shown in Figure 1.



CitizenId	FirstName	LastName	DateOfBirth	Sex
1004	Tonye	Alalibo	2002-08-28	M
1003	Bari	Konyaa	1983-08-16	F
1003	Bari	Konyaa	1943-08-16	F
1103	Amaka	Didia	1978-06-07	F
1107	Tonye	Tamuno	1990-10-16	F

1102	Achinike	Ihunda	1969-12-13	M
1108	Umoh	Okon	1991-08-20	M
1113	Femi	Abiola	1981-03-03	M
1118	Furo	Amadi	1985-04-05	M

ChildBirthId	DateOfEvent	PlaceOfEvent	ExperiencedBy
2001	2023-01-06	Bidere	1003
2002	2023-01-13	Abonnema	1107
2003	2023-02-01	Elele	1103
2004	2019-04-13	Port Harcourt	1003

DateIssued	AssociatedWith	BabyFather
2023-01-07	2001	1102
2023-01-13	2002	1118
2023-02-01	2003	1113
2019-04-15	2004	1108

Figure 1: Child Birth Data Model and Fragment Dataset

For direct mapping, the input to the process is the dataset and schema and the output is the translated RDF data. Essentially, RDB2RDF direct mapping specification is an algorithm to carry out the conversion. Each data row is viewed as a set of triples describing an entity. A URI resource is generated for each row in the citizen table, which has a primary key defined on the CitizenID column. Given a predefined URI base http://webgov.riversstate.gov.ng/RV/, a URI resource of http://webgov.riversstate.gov.ng/RV/Citizen/CitizenID = 1003 is generated for the first row (CitizenID = 1003) of the Citizen table, following the syntax in the RDB2RDF specification. The syntax is in the form:

'URI_BASE' + COLUMN_NAME = COLUMN_VALUE

Where URI_BASE is the URI prefix and COLUMN is the column in the data row of each column in the row.

A blank node is generated for each of tables without a primary key. If the Birth_Reg table is created without a primary key, the first row can be represented with a blank node of _:bn1, to uniquely represent the row. The next step in the algorithm is to convert the data row into RDF triples describing the generated resources. A basic triple to be generated is the triple asserting the type of the data row resource as an instance of its table class. The type assertion of the Citizen table's first row is:

<RV:Citizen/CitizenID = 1003, rdf:type, RV:Citizen>

where DB is the URI_BASE

In addition to the type assertion, a data_valued triple with literal values, as the object, is generated for columns that have no foreign key definition. For columns with foreign key, the generated triples are relational, and their objects are either URI resources or blank nodes. For the Citizen table, there is no foreign key defined for the FirstName value of its 'Bari' data row is generated as the triple:

<RV:Citizen/CitizenID = 1003, RV:Citizen#FirstName, 'Bari'>

For the foreign key columns, the main aspect considered in generating the triples, is how to get the RDF resource of the object in the triple. Since a foreign key is referencing the other data row, in another table, the entity denoted by the foreign key column(s) should be generated from the referenced data row accordingly. The ExperiencedBy column, in the Child_Birth table, is specified as a foreign key referencing the CitizenID in the Citizen table. The column value will be converted into a relation between the '2001' child_birth and citizen 'Bari'. The entity denoted by ExperiencedBy value '1003' need to be generated from the third row of the citizen table, which is:

RV:Citizen/CitizenID = 1003

So, the triple to be generated from the ExperiencedBy column is:

RV: Child_Birth/Child_BirthID = 2001, RV: Child_Birth#ref – ExperiencedBy,
RV:Citizen/CitizenID = 1003

Applying the foregoing logic on all data tables, the RDF triples from our WebGov database is generated.

3.2 RDB2RDF Mapping Language (R2RML)

While direct mapping is an efficient way to quickly realize RDF triples, it does not afford the flexibility needed to obtain desirable characteristics in the generated triples. One such desirable properties is the desire to use popular domain ontologies such as FOAF ontology [14] or the person ontology in DBPedia or Schema.org, to improve the visibility of a knowledge base and make it easier to integrate with other knowledge bases. Second, direct mapping creates blank nodes in table without a primary key such as the Birth_Reg table. A mapping language allows one to use custom properties to properly specify the relationships between the entities in the table. Third, in some situations, one may need to hide confidential information, for example hiding some personal information of a citizen.

A customized conversion, realized by an RDB2RDF mapping language, is needed to realize all the aforementioned desirable characteristics. Realizing the first characteristics requires customized RDF resource generation; the second characteristics needs the customization of the mapping; and the third characteristics requires the ability to extract part of the data for conversion. These requirements correspond to the constructs of the RDB2RDF Mapping Language – Term Maps, Logical Tables, and Triple Maps.

A Term Map is a function used in generating an RDF resource from data rows. For the Citizen table in our dataset, generating a type assertion for each row using a preferred DBPedia citizen vocabulary (dbo:Citizen) instead of the cryptic vocabulary generated by direct mapping, uses the following mapping definition:

[ ]   rr: predicateMap [rr: constant rdf: type];

rr: objectMap [rr: constant dbo: Citizen]

It is important to note that, we use dbo:Citizen vocabulary only to illustrate the importance of using a publicly available vocabulary, it does not exist in this form. The closest concept to citizen that exist in DBPedia is dbo:Citizenship.

The 'constant' in the mapping indicates that for every row in the citizen table, the mapping generates the same pair of RDF resources – rdf:type and dbo:citizen, for the predicate and object of the generated assertion respectively.

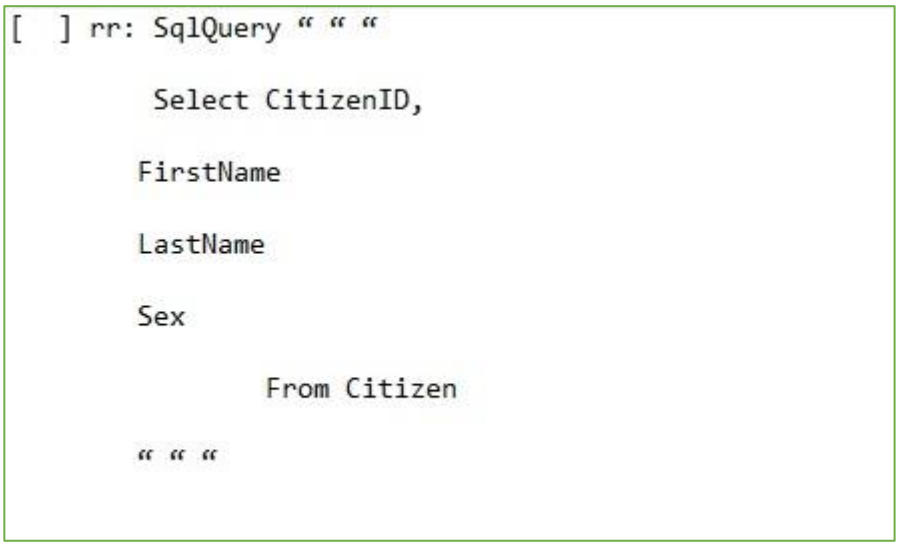
In addition to the constant-value term map for generating RDF resources for each row, a column-value term map is used to generate a literal resource for a data column. For our Citizen table, the following mapping generates a literal resource as the object of a triple using the value of the 'firstname' column:

[ ] rr:objectMap [rr: column "FirstName"]

A template-valued term map is used to design a customized URI scheme using a string template. The following mapping defines our customized Citizen URIs using the CitizenID column as the variable part:

[ ] rr: subjecMap [rr: template http://webgov.riversstate.gov.ng/RV/Citizen/ID/{CitizenID}]

Logical tables, in R2RML, is a way to enable the customized data extraction and transformation from the original database using SQL queries. For the Citizen table, one may want to hide the Date_of_Birth information in its RDF version; the following logical table is defined using a simple SQL query to select necessary columns only, as shown in figure 2:



```
[   ] rr: SqlQuery " " "

         Select CitizenID,

        FirstName

        LastName

        Sex

                From Citizen

        " " "
```

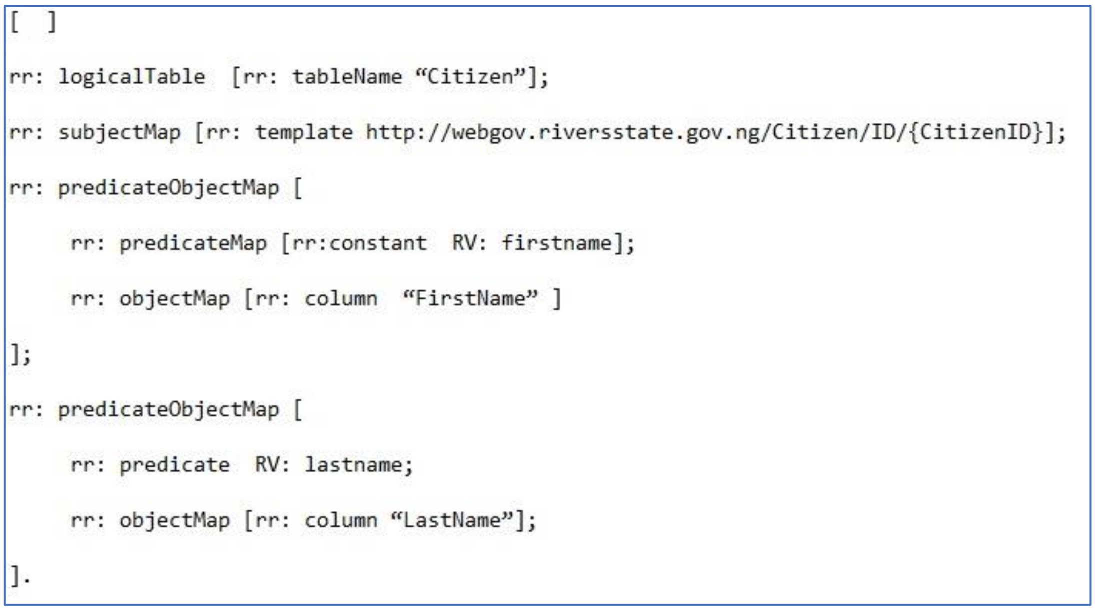
Figure 2: Logical table from SQL Query for Citizen Table

With the use of SQL queries in logical tables, comes the ability to carry out data transformation, including using the SQL built-in function such as MD5 for data transformation. The simplest form of logical table is the direct use of a table or view defined in a database:

[ ] rr: tableName "Citizen"

A triple map brings all components of the mapping definition together in one transformation. It provides a complete specification of how a data row is converted into a set of RDF triples.

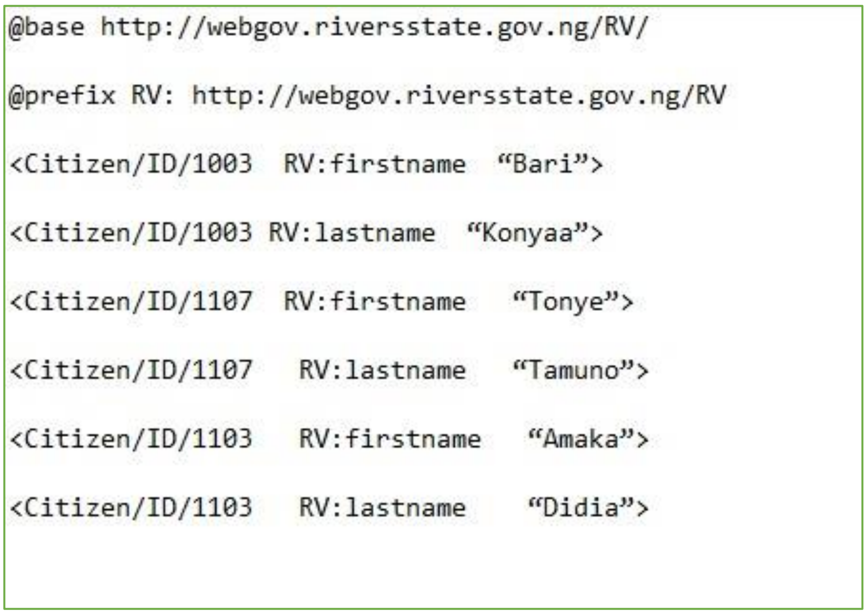
For the Citizen table, the definition of the rules to convert the Firstname and LastName columns into two RDF triples is given in figure 3 as follows:



```
[  ]

rr: logicalTable  [rr: tableName "Citizen"];

rr: subjectMap [rr: template http://webgov.riversstate.gov.ng/Citizen/ID/{CitizenID}];

rr: predicateObjectMap [

     rr: predicateMap [rr:constant  RV: firstname];

     rr: objectMap [rr: column  "FirstName" ]

];

rr: predicateObjectMap [

     rr: predicate  RV: lastname;

     rr: objectMap [rr: column "LastName"];

].
```

Figure 3: A Triple Map of the Citizen Table

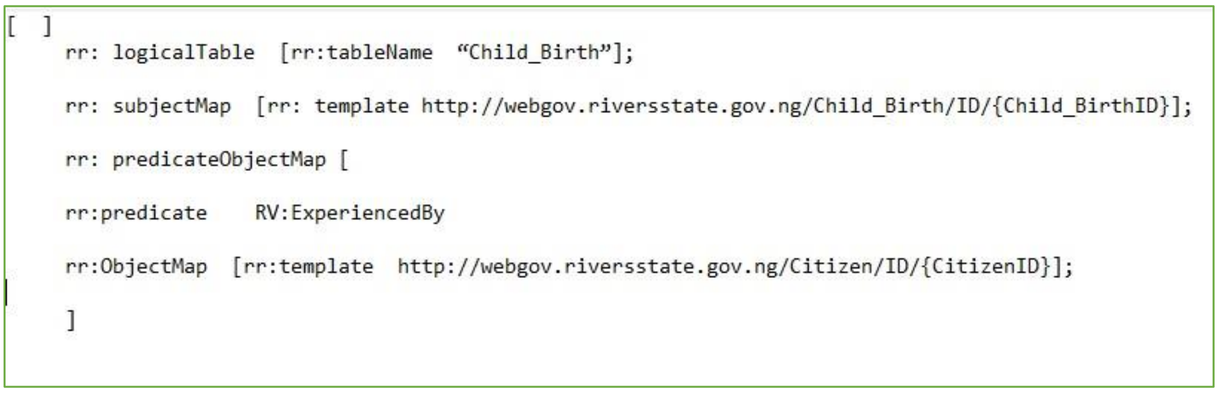
A triple map consists of one logical table, one subject map, one or more predicateObjectMaps. In the Citizen table mapping, there are two predicateObjectMaps, each defining a triple to be generated on a column-valued term map. The following triples, in figure 4, are generated from the triple map above:



```
@base http://webgov.riversstate.gov.ng/RV/

@prefix RV: http://webgov.riversstate.gov.ng/RV

<Citizen/ID/1003  RV:firstname  “Bari”>

<Citizen/ID/1003 RV:lastname  “Konyaa”>

<Citizen/ID/1107  RV:firstname   “Tonye”>

<Citizen/ID/1107   RV:lastname   “Tamuno”>

<Citizen/ID/1103   RV:firstname   “Amaka”>

<Citizen/ID/1103   RV:lastname    “Didia”>
```

Figure 4: A Triples Map Output for the Citizen Table

So far, the mappings specified have not captured cases in which foreign keys define relations between two or more tables. Relational triples can be generated between the Child_Birth and Citizen tables in the following definition in figure 5:



```
[ ]
    rr: logicalTable  [rr:tableName  “Child_Birth”];

    rr: subjectMap  [rr: template http://webgov.riversstate.gov.ng/Child_Birth/ID/{Child_BirthID}];

    rr: predicateObjectMap [

    rr:predicate    RV:ExperiencedBy

    rr:ObjectMap  [rr:template  http://webgov.riversstate.gov.ng/Citizen/ID/{CitizenID}];

    ]
```

Figure 5: Relational Triple of the Child_Birth and Citizens Tables

RDF Mapping Language (RML) [15] is an extension of R2RDF, and it is used to declare rules on how to generate RDF files. A key feature of RML is that in addition to relational data sources, it also processes flat files, such as JSON and CSV files as data input.

Writing a mapping to transform a reasonable size data is tedious, mainly due to the fact that the mapping files are designed for machine processing, as opposed to human processing. In order to aid developer productivity, RML include as part of a toolset, YARRRML. YARRRML [16] is a human readable text-based representation for declarative Linked Data generation, and it is a subset of YAML [17] – a data serialization language, used mainly for configuration files and in applications where data is being stored or transmitted. Rules are defined within a YAML file, which are then transformed to RML or R2RML.

Figure 6 shows, in UML activity diagram, the steps involved in the data transformation using RML and an RDB data source.

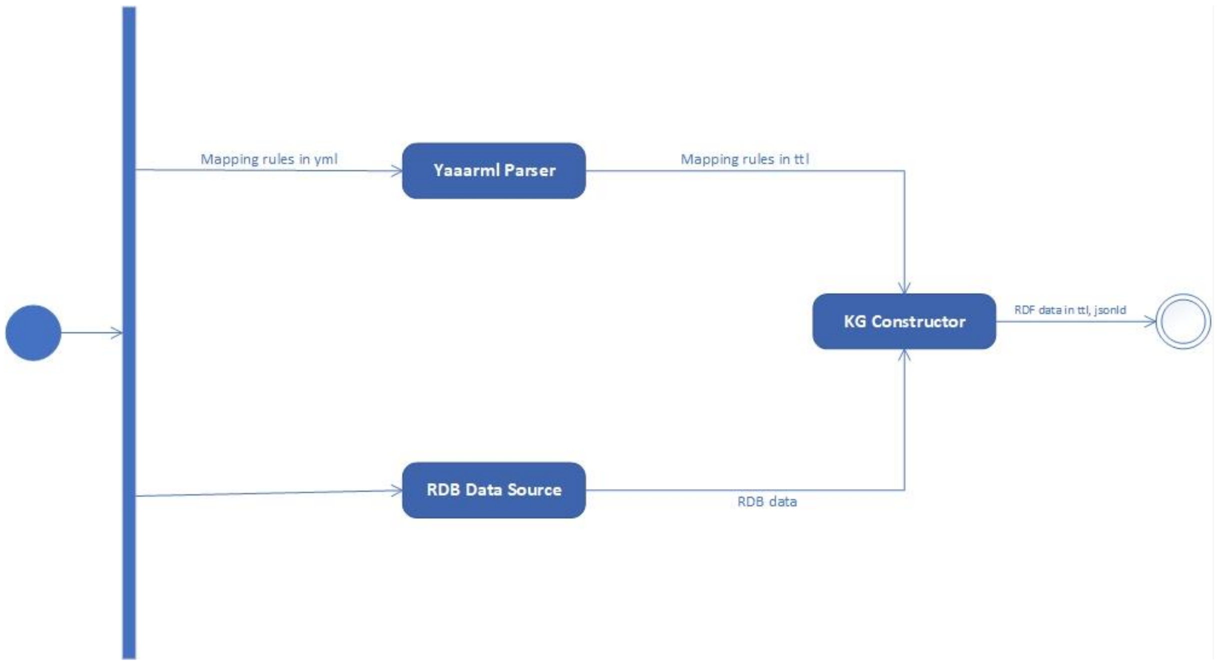
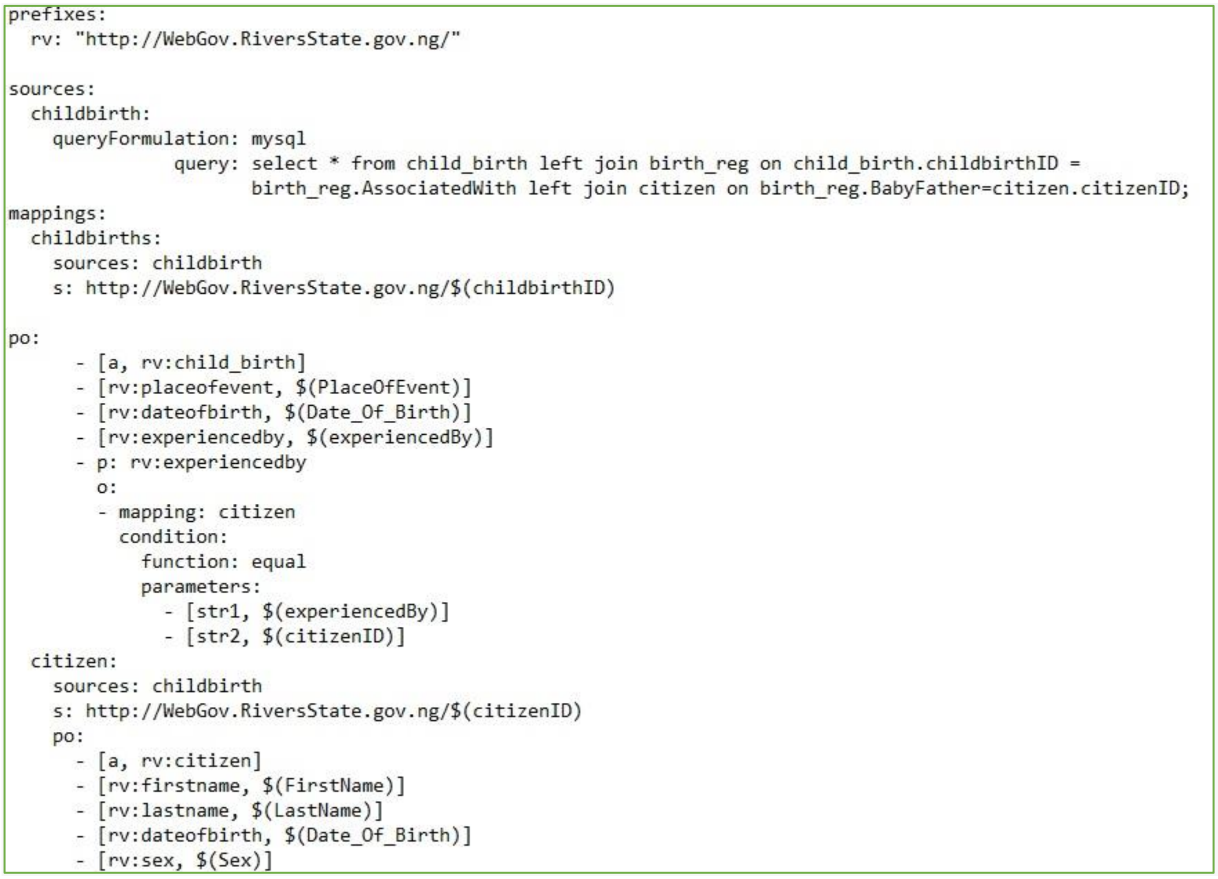


Figure 6: UML Activity diagram Showing the Steps in the RML Data Transformation

Data transform mapping rules are first written in a yml file. The yml file is then used as an input to the yaaarml parser [18], which produces mapping rules in ttl file format. The KG constructor then produces the RDF triples, using the mapping rules file and the RDB data as input. The KG constructor output is in ttl and json-ld formats. The KG constructor used in this model is Morph-KGC [19], an engine that constructs RDF KG from heterogeneous data sources with R2RML and RML mapping languages.

The entire webgov database can be accessed with the "sources" section of a yml file, where the database tables and SQL queries in query Formulation are declared. In addition to the "sources" definition, a YAML file also contain definitions for "prefixes" and "mapping" as shown in our webgov_rules file as shown in Figure 7:



```
prefixes:
  rv: "http://WebGov.RiversState.gov.ng/"

sources:
  childbirth:
    queryFormulation: mysql
               query: select * from child_birth left join birth_reg on child_birth.childbirthID =
                      birth_reg.AssociatedWith left join citizen on birth_reg.BabyFather=citizen.citizenID;
mappings:
  childbirths:
    sources: childbirth
    s: http://WebGov.RiversState.gov.ng/$(childbirthID)

po:
      - [a, rv:child_birth]
      - [rv:placeofevent, $(PlaceOfEvent)]
      - [rv:dateofbirth, $(Date_Of_Birth)]
      - [rv:experiencedby, $(experiencedBy)]
      - p: rv:experiencedby
        o:
        - mapping: citizen
          condition:
            function: equal
            parameters:
              - [str1, $(experiencedBy)]
              - [str2, $(citizenID)]
  citizen:
    sources: childbirth
    s: http://WebGov.RiversState.gov.ng/$(citizenID)
    po:
      - [a, rv:citizen]
      - [rv:firstname, $(FirstName)]
      - [rv:lastname, $(LastName)]
      - [rv:dateofbirth, $(Date_Of_Birth)]
      - [rv:sex, $(Sex)]
```

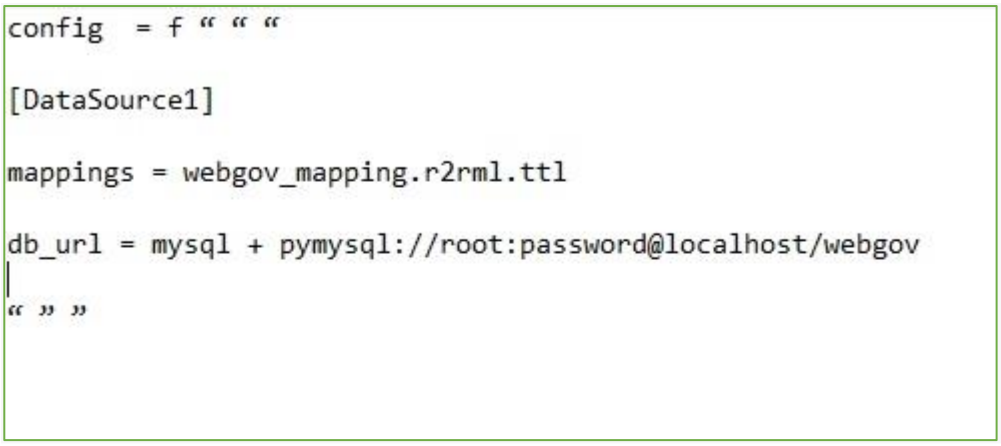
Figure 7: A WebGov Mapping Rules

A command-line yarrrml parser is used to transform the webgov_rules.yml file to R2RML:

yarrrml -parser -i webgov_rules.yml -o webgov_rules_mapping.r2rml.ttl -f

As stated, a mapping file contain rules for transforming a dataset from RDB to RDF. So, the input to this transformation is the RDB dataset and the mapping file obtained from yarrrml parser above, as well as an algorithm for transforming the dataset based on the mapping. This algorithm can be written using any programming language and it is also captured in many tools used in R2RML. We use Kglab, a tool that provides a simple abstraction layer in Python for building KGs. Kglab integrates with popular graph libraries, including RDFlib, OWL-RL, pySHACL, NetworkX, iGraph, PyVis, node2vec, pslpython, pgmpy and other data science libarries such as Pandas, Numpy, scikit-learn, matplotlib and PyTorch.

One Python library that is included as part of the Kglab package is Morph-kgc. Morph-kgc is an engine that constructs RDF KGs from heterogeneous data sources using R2RML. Morph-kgc uses a configuration string, in Python, to describe the mapping. A configuration string links the mapping file in Python, and the path and access to the datasource, as shown in Figure 8.

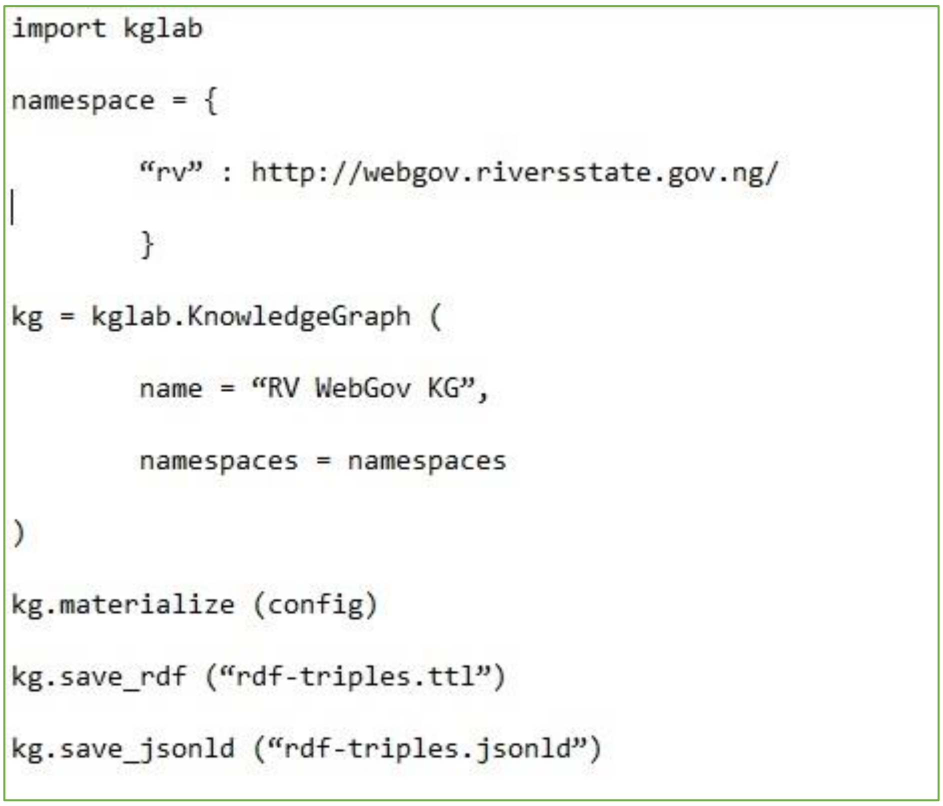


```
config  = f " " "

[DataSource1]

mappings = webgov_mapping.r2rml.ttl

db_url = mysql + pymysql://root:password@localhost/webgov

" " "
```

Figure 8: Configuration for RDF Generation in Python

This configuration references a R2RML mapping (in webgov_mapping.r2rml.ttl), which gets applied to the input data – webgov database in a MySQL DBMS.

We then use the Morph-kgc to load the RDF data from the MySql datasource based on an R2RML mapping as shown in Figure 9.

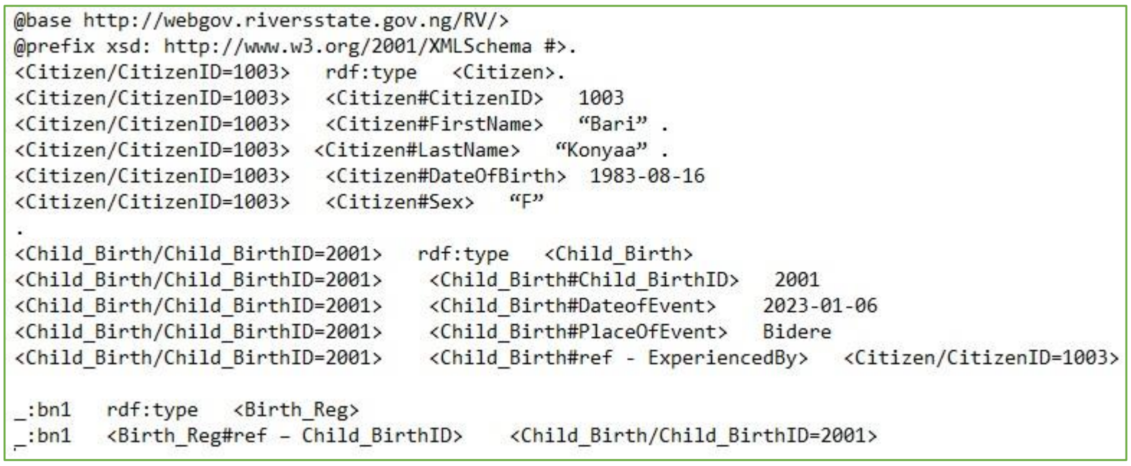


```
import kglab

namespace = {

        "rv" : http://webgov.riversstate.gov.ng/

        }

kg = kglab.KnowledgeGraph (

        name = "RV WebGov KG",

        namespaces = namespaces

)

kg.materialize (config)

kg.save_rdf ("rdf-triples.ttl")

kg.save_jsonld ("rdf-triples.jsonld")
```

Figure 9: An RDF data Generation in Python

4. Results and Discussion

Using a base IRI – *http://webgov.riversstate.gov.ng/RV/*, direct mapping of a fragment of the webgov database produces the direct graph shown in Figure 10.

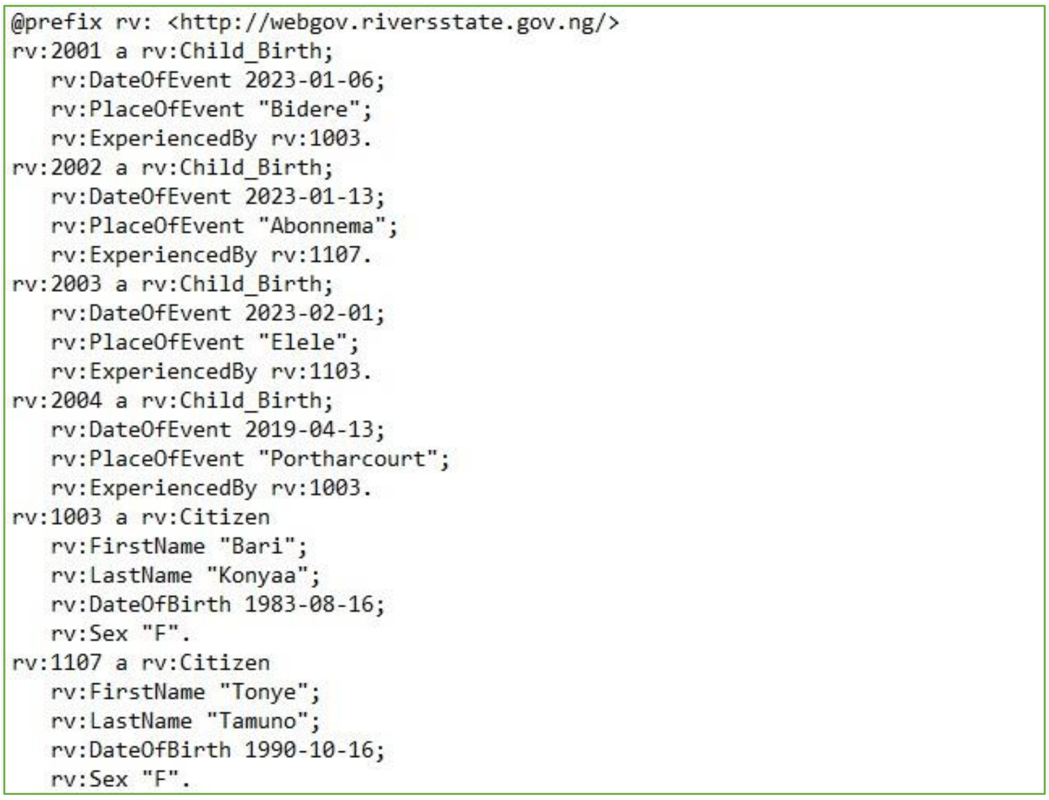


```
@base http://webgov.riversstate.gov.ng/RV/>
@prefix xsd: http://www.w3.org/2001/XMLSchema #>.
<Citizen/CitizenID=1003>   rdf:type   <Citizen>.
<Citizen/CitizenID=1003>   <Citizen#CitizenID>   1003
<Citizen/CitizenID=1003>   <Citizen#FirstName>   "Bari" .
<Citizen/CitizenID=1003>  <Citizen#LastName>   "Konyaa" .
<Citizen/CitizenID=1003>   <Citizen#DateOfBirth>  1983-08-16
<Citizen/CitizenID=1003>   <Citizen#Sex>   "F"
.
<Child_Birth/Child_BirthID=2001>   rdf:type   <Child_Birth>
<Child_Birth/Child_BirthID=2001>    <Child_Birth#Child_BirthID>   2001
<Child_Birth/Child_BirthID=2001>    <Child_Birth#DateofEvent>    2023-01-06
<Child_Birth/Child_BirthID=2001>    <Child_Birth#PlaceOfEvent>   Bidere
<Child_Birth/Child_BirthID=2001>    <Child_Birth#ref - ExperiencedBy>   <Citizen/CitizenID=1003>

_:bn1   rdf:type   <Birth_Reg>
_:bn1   <Birth_Reg#ref - Child_BirthID>    <Child_Birth/Child_BirthID=2001>
```

Figure 10: Fragment of a Direct Graph of the Webgov Database

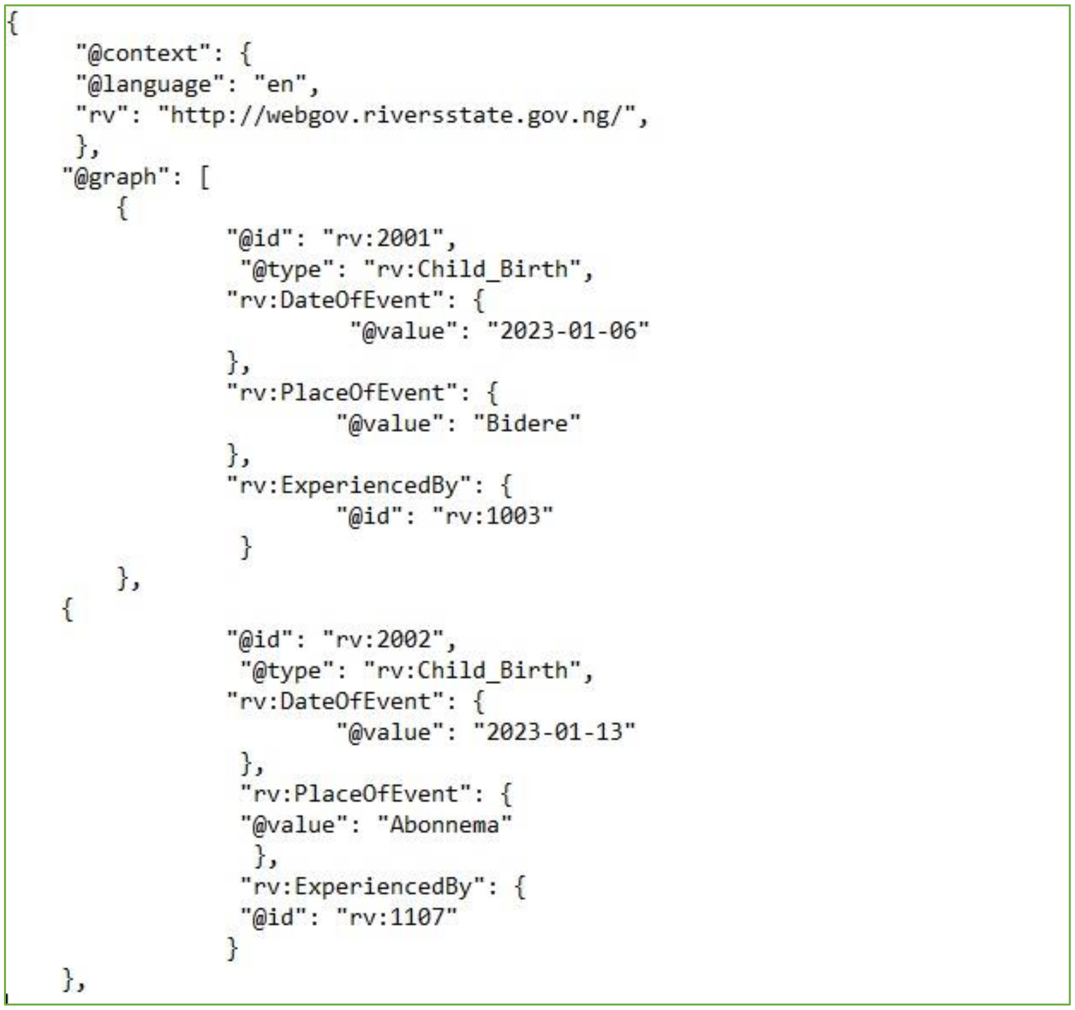
In the fragment shown in figure 4.3, each row of the Citizen table produces a set of triples with a common subject. The subject is an IRI formed from the concateneation of the base IRI, the Citizen table name, primary key column name (CitizenID) and the primary key value (1003). The predicate for each column is an IRI for devised from the concatenation of the base IRI, the table name (Citizen) and the column name (e.g FirstName). The objects are RDF literals created from the column value. The same mapping schema is followed for the Child_Birth table. The foreign key in the Child_Birth table produces a triple with a predicate composed from the foreign key column name (ExperiencedBy), the referenced table (Citizen), and the referenced column name (CitizenID = 1003). This make the object of the triple to be row identifier (Citizen/CitizenID = 1003) for the referenced triple. Notice that these reference row identifiers must coincide with the subject used for the triples generated from the referenced row.

Data transformation using a mapping language - RML makes use of tools that generate intermediate results in the process of generating the RDF graph output. As stated earlier, using RML, affords the flexibility to customize the generated graph and provides ease of mapping with the use of human-readable mapping constructs. The output graph shown in figure 11 is the generated output using RML. Notice that the output is similar to the output with direct mapping for table without a foreign key. For those tables with a foreien key, the generated output graph show a marked difference depending on the customization needed in the output. On additional customization provided by using a mapping languge is that the output graph can be provided in different file format. For example, the output of the graph in figure 11 is generated in jsonld file format as shown in Figure 12.



```
@prefix rv: <http://webgov.riversstate.gov.ng/>
rv:2001 a rv:Child_Birth;
   rv:DateOfEvent 2023-01-06;
   rv:PlaceOfEvent "Bidere";
   rv:ExperiencedBy rv:1003.
rv:2002 a rv:Child_Birth;
   rv:DateOfEvent 2023-01-13;
   rv:PlaceOfEvent "Abonnema";
   rv:ExperiencedBy rv:1107.
rv:2003 a rv:Child_Birth;
   rv:DateOfEvent 2023-02-01;
   rv:PlaceOfEvent "Elele";
   rv:ExperiencedBy rv:1103.
rv:2004 a rv:Child_Birth;
   rv:DateOfEvent 2019-04-13;
   rv:PlaceOfEvent "Portharcourt";
   rv:ExperiencedBy rv:1003.
rv:1003 a rv:Citizen
   rv:FirstName "Bari";
   rv:LastName "Konyaa";
   rv:DateOfBirth 1983-08-16;
   rv:Sex "F".
rv:1107 a rv:Citizen
   rv:FirstName "Tonye";
   rv:LastName "Tamuno";
   rv:DateOfBirth 1990-10-16;
   rv:Sex "F".
```

Figure 11: Fragment of the Mapping Output in Turtle Format



```
{
    "@context": {
    "@language": "en",
    "rv": "http://webgov.riversstate.gov.ng/",
    },
   "@graph": [
       {
               "@id": "rv:2001",
                "@type": "rv:Child_Birth",
               "rv:DateOfEvent": {
                        "@value": "2023-01-06"
               },
               "rv:PlaceOfEvent": {
                       "@value": "Bidere"
               },
               "rv:ExperiencedBy": {
                       "@id": "rv:1003"
                }
       },
   {
               "@id": "rv:2002",
                "@type": "rv:Child_Birth",
               "rv:DateOfEvent": {
                       "@value": "2023-01-13"
                },
                "rv:PlaceOfEvent": {
                "@value": "Abonnema"
                 },
                "rv:ExperiencedBy": {
                "@id": "rv:1107"
               }
   },
```

Figure 12: Fragment of the RML Output in jsonld Format

Input Data Variation

The data tranformation model was fed with variations of the input data the see whether a data transformation will be carried and also see what the nature of the output will be. Accordingly the input data was altered according the following test case criteria in Table 1:

Table 1: Data Transform Input Data Variation

1.	Standard case with valid query in mapping rules

2	Invalid Query in mapping rules
3	Missing Query in mapping rules
4	Entity not in DB (wrong information)
5	Empty Table
6	Standard case with configuration file
7	Missing source section in mapping rules
8	Missing configuration in transform command
9	Invalid mapping rules
10	Missing PyMySQL package

The results in Table 2 was obtained for the input data variation experiment:

Table 2: Results from the Input Data Variation Experiment

	Test Cases	Output
1	Standard case with valid query in mapping rules	Generated RDF graph
2	Invalid Query in mapping rules	No generated RDF graph
3	Missing Query in mapping rules	No generated RDF graph
4	Entity not in DB (wrong information)	No generated RDF graph
5	Empty Table	No generated RDF graph
6	Standard case with configuration file	Generated RDF graph
7	Missing source section in mapping rules	No generated RDF graph
8	Missing configuration in transform command	No generated RDF graph
9	Invalid mapping rules	No generated RDF graph
10	Missing PyMySQL package	No generated RDF graph

The RDB2RDF data transformation task is evaluated against the W3C requirements for the features of R2RML [20], and these features are summarized in Table 3.

In addition, Table 2 shows that, in the data transformation experiments, eight cases of the 10 experiments carried-out, the model did not return a tranformation. This is because the transformation model is designed to produce the RDF data output only if the input data is in the accepted form, and the mapping rules has the coverage of all the component entities of the source data.

Table 3: Features of Customized Mapping

Feature	Description
Generation of user defined IDs	Ability to generate URIs of resources beyond the simple use of primary key values: reusing and combining column values, allowing for conversion tables, etc.
Logical table	Ability to read tuples not only from tables but also from SQL views or from the result of an SQL query.
Column selection	Ability to select only a subset of the columns of a table to translate. This is a very basic feature, almost a minimum pre-requisite of any RDB-to-RDF tool.
Column renaming	Ability to map a column to an RDF property with a different name. This is not always possible in a direct mapping but quite obvious in a domain semantics-driven mapping.
Vocabulary reuse	Ability to map relational entities to instances of existing vocabularies and ontologies. This is the main difference between domain semantics-driven mapping and direct mapping approaches.
1 table to n classes	Ability to use the values of a column as a categorization pattern: tuples of the table will be translated into instances of different ontological classes based on the value of this attribute. This feature can be seen as an extension of the "select conditions" feature as it results in not only filtering out rows, but the filter helps selecting rows to be converted into instance of one class or another.

Many-to-many relation to simple triples	Many-to-many relations are usually implemented in relational databases as a join table which columns are all foreign keys to other tables (n-ary relations). This feature consists of the ability to translate many-to-many join tables into simple triples. This opposes to a basic direct mapping in which the join table will be translated into a distinct class.
Blank nodes	Ability to generate blank nodes and refer to them within the graph produced during the translation process. Blank nodes can be used for instance to translate a table without a primary key.
Data types	Ability to handle relational data types consistently with RDF data types per SQL-XSD mapping.
Data transformation	Ability to apply transformation functions to the values before generating the RDF triples. This can be used to perform complex type conversion, compute a value using several columns, and applying methods such as string manipulation functions, decimals type conversions, etc.
Named graphs	Ability to create not only a default RDF graph but also multiple named graphs within a single mapping definition.
User-defined namespaces	Ability to declare and use namespace prefixes.
Static metadata	Ability to attach static metadata (such as licensing or provenance information) to the produced graphs, and possibly to all RDF entities or instances of a certain class.

5. Conclusion

In this paper, we developed a model for transforming data from relational to RDF in an e-Government context. Our approach is one of two types, based on whether the transformation needed is direct, in which case, little or no control is exercised on the transformation output. If more control is required on the transformation output, then RML transformation language is used. Our approach relies on using a yml parser to convert mapping rule in yml form to ttl form, which is then used to transform the input data to ttl or json form.

Our transformation model is designed to produce the RDF data output only if the input data is in the accepted form, and the mapping rules has the coverage of all the component entities of the source data. However, limitations exist in our model. Our model experiments are carried out within the context of a specific e-Government context, and the dataset used are synthetic. Results obtained should generally apply to other e-Government contexts and generally reflect results obtained with real data. However, generalizability is not the main focus of our work. As future work, we intend to extend our model to address these limitations, and accommodate more diverse dataset.

REFERENCE

[1] A. Adadi, M. Berrada, D. Chenouni, and B. Bounabat, "Ontology based composition of e-Government services using AI Planning," in *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, Rabat: IEEE, Oct. 2015, pp. 1–8. doi: 10.1109/SITA.2015.7358430.

[2] A. Adadi, M. Berrada, and N. El Akkad, "Artificial Intelligence based Composition for E-Government Services," in *Proceedings of the Third International Conference on Computing and Wireless Communication Systems, ICCWCS 2019, April 24-25, 2019, Faculty of Sciences, Ibn Tofaïl University -Kénitra- Morocco*, Kenitra, Morocco: EAI, 2019. doi: 10.4108/eai.24-4-2019.2284071.

[3] A. Adadi, M. Berrada, D. Chenouni, and B. Bounabat, "A SEMANTIC WEB SERVICE COMPOSITION FOR E- GOVERNMENT SERVICES," . *Vol.*, p. 8, 2015.

[4] O. S. Al-Mushayt, "Automating E-Government Services With Artificial Intelligence," *IEEE Access*, vol. 7, pp. 146821–146829, 2019, doi: 10.1109/ACCESS.2019.2946204.

[5] IEEE Computer Society e-Gov STC, "About - IEEE Computer Society e-Government STC," About - IEEE Computer Society e-Government STC. Accessed: Nov. 25, 2022. [Online]. Available: https://sites.google.com/a/ieee.net/stc-egov/about

[6] D. Fensel *et al.*, *Knowledge graphs*. Springer, 2020.

[7] M. Chatzidimitriou and A. Koumpis, "Marketing One-stop e-Government Solutions: the European OneStopGov Project," *IAENG Int. J. Comput. Sci.*, vol. 35, no. 1, p. 7, 2008.

[8] E. Hovy, "Data and knowledge integration for e-government," in *Digital government: E-Government research, case studies, and implementation*, Springer, 2008, pp. 219–231.

[9] J. Sequeda and O. Lassila, *Designing and Building Enterprise Knowledge Graphs*. in Synthesis Lectures on Data, Semantics, and Knowledge. Cham: Springer International Publishing, 2021. doi: 10.1007/978-3-031-01916-6.

[10] J. Cowie and W. Lehnert, "Information extraction," *Commun. ACM*, vol. 39, no. 1, pp. 80–91, 1996.

[11] Das, Souripriya, Sundara, Seema, and Cyganiak, Richard, "R2RML: RDB to RDF Mapping Language." Sep. 27, 2012. Accessed: Nov. 12, 2023. [Online]. Available: https://www.w3.org/TR/r2rml/

[12] Connolly, Dan, "Gleaning Resource Descriptions from Dialects of Languages (GRDDL)." Sep. 11, 2007. Accessed: Nov. 12, 2023. [Online]. Available: https://www.w3.org/TR/grddl/

[13] Arenas, Marcelo, Bertails, Alexandre, Prud'hommeaux, Eric, and Sequeda, Juan, "A Direct Mapping of Relational Data to RDF." 27December2012. Accessed: Nov. 30, 2023. [Online]. Available: https://www.w3.org/TR/rdb-direct-mapping/

[14] Brickley, Dan and Miller, Libby, "FOAF Vocabulary Specification." 1May2004. Accessed: Dec. 02, 2023. [Online]. Available: http://xmlns.com/foaf/0.1/

[15] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle, “RML: A generic language for integrated RDF mappings of heterogeneous data.,” *Ldow*, vol. 1184, 2014.

[16] Van Assche, Dylan, De Meester, Ben, Heyvaert, Pieter, and Dimou, Anastacia, “YARRRML.” 26January2023. Accessed: Nov. 25, 2023. [Online]. Available: https://rml.io/yarrrml/spec/

[17] O. Ben-Kiki, C. Evans, and B. Ingerson, “Yaml ain’t markup language (yaml^{TM}) version 1.1,” *Work. Draft 2008*, vol. 5, p. 11, 2009.

[18] “YARRRML Parser.” RDF Mapping Language (RML), Aug. 18, 2023. Accessed: Nov. 25, 2023. [Online]. Available: https://github.com/RMLio/yarrrml-parser

[19] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M. S. Pérez, and O. Corcho, “Morph-KGC: Scalable knowledge graph materialization with mapping partitions,” *Semantic Web*, pp. 1–20, Aug. 2022, doi: 10.3233/SW-223135.

[20] Auer, Soren, Feigenbaum, Lee, Miranker, Daniel, Fogarolli, Angela, and Sequeda, Juan, “Use Cases and Requirements for Mapping Relational Databases to RDF.” 8June2010. Accessed: Nov. 22, 2023. [Online]. Available: https://www.w3.org/TR/rdb2rdf-ucr/