ARDI: Automatic Generation of RDFS Models from Heterogeneous Data Sources

Shumet Tadesse*, Cristina Gómez*, Oscar Romero*, Katja Hose†, and Kashif Rabbani*
* Dept. of Service and Information System Engineering
Universitat Politecnica de Catalunya, BarcelonaTech Barcelona, Spain
{*shumet, cristina, oromero, krabbani*}*@essi.upc.edu*
†Department of Computer Science
Aalborg University, Aalborg, Denmark
khose@cs.aau.dk

***Abstract*—The current wealth of information, typically known as Big Data, generates a large amount of available data for organisations. Data Integration provides foundations to query disparate data sources as if they were integrated into a single source. However, current data integration tools are far from being useful for most organisations due to the heterogeneous nature of data sources, which represents a challenge for current frameworks. To enable data integration of highly heterogeneous and disparate data sources, this paper proposes a method to extract the schema from semi-structured (such as JSON and XML) and structured (such as relational) data sources, and generate an equivalent RDFS representation. The output of our method complements current frameworks and reduces the manual workload required to represent the input data sources in terms of the integration canonical data model. Our approach proposes a set of production rules at the meta-model level to ensure that the model translations are correct. Finally, a tool for implementing our approach has been developed.**

***Index Terms*—Data Model Translation, Data Integration, RDF Schema, Meta-modeling**

I. Introduction

In today's digital world, data, so-called Big Data, are one of the organizations most important strategic assets. Organizations envisage this asset to achieve business success by improving client experience, improving working procedures, and increasing efficiency. On the one hand, most organizations use *agility* as an important business strategy to deliver better services [33]. Relevant data sources for these organizations reside uncoordinated across different departments or even in external sources. As a result, organizations must be able to integrate their data to drive their business from such fragmented data. For example, interaction data that is massively generated from Social Networks (such as Twitter, Facebook) can be integrated with organizational data to get an enhanced view of their product diffusion.

Big Data is mainly characterized by *volume*, *variety* and *velocity* [1], [19]. The size of data can range from terabytes to zettabytes or beyond and data may be produced in different formats, i.e., structured, semi-structured or unstructured. Big Data sources typically come in terms of schemaless data models such as XML or JSON (e.g., API calls, document-stores such as MongoDB, etc.). It is also essential to include the available organizational data, typically stored in a relational database. The relational model is a structured data model that facilitates the translation to other languages (i.e., every relational database contains a catalog with the necessary structural meta-data or, in database terms, the schema information). However, for schemaless data formats (e.g., JSON or XML) there is typically no such available meta-data (e.g., MongoDB, CouchDB or ElasticSearch do not either support JSON schema or real databases do not use this feature). Such characteristics represent a challenge for users to get the desired data potentially spanning multiple independent data sources. As identified by the Sloan Review Group[1], the current focus of Big Data is dealing with data integration in such settings.

Data integration is a well-studied field that provides unified access to disparate data sources by means of a unified view expressed in terms of a canonical model [14]. However, it is well-known that traditional data integration techniques cannot be used as-is for Big Data Integration [13]. The main steps during data integration are: expressing each data source in terms of the chosen canonical data model, creating a single unified view of the sources (also known as integration or target schema) and mapping the data sources to the target schema. Here **we focus on the very first step towards integrating heterogeneous data sources** (i.e. choosing a canonical data model and translating the scheme of data sources into the chosen canonical data model). In this scenario, the Semantic Web (SW) data models become a key technology for data standardization and conceptualization. As stated in [10], recent trends show that data from different sources are migrated to semantic technologies to facilitate data integration. For example, ontology-based frameworks for data integration like [5] and [25] involve the use of SW data models such as RDF/RDFS or OWL to combine heterogeneous data sources effectively.

From the viewpoint of modeling, and unlike OWL, RDF Schema (RDFS) is capable of expressing resources in a structured way separated into three layers (i.e., instance, model and meta-model). Such modeling is achieved through *rdf:type*, a property used to distinguish instances from classes. Other properties, such as *rdfs:domain* and *rdfs:range* allow to express

[1] http://bit.ly/TheVarietyChallenge

 DOI 10.1109/EDOC.2019.00031

additional constraints at the model level. Meta-modeling is used in software engineering to construct models of models [18], which has been vastly overlooked in other fields. The use of MDA (Model Driven Architecture) principles in the development of SW ontologies is both desirable and promising [11]. It enhances agility in the development of SW applications. To promote the best practices of MDA approaches in the SW community, Object Management Group (OMG) [27] provides detailed specifications for RDFS and OWL metamodels.

In order to enable meta-modeling, we choose RDFS as a canonical data model, since we argue that meta-modeling can be the solution for Big Data Integration. A significant advantage of meta-modeling is the capability of supporting different abstraction levels. It helps to maximize the extent to which data can be integrated by separately expressing meta-schema, schema information and the data itself. In addition, it ensures semantic interoperability. From a technical point of view, meta-modeling approaches help to minimize development time and maximize efficiency and productivity [7].

In this paper, we present a generic approach that overcomes the limitations of the current state of the art for extracting and representing disparate data sources in terms of the canonical data model. As such, **this paper fosters the use of meta-modeling in automating the steps of translating heterogeneous data sources to a common data model**. Our approach not only automates the extraction and translation of source schemas to RDFS (the integration canonical data model), but **it is the first approach providing foundations to generate valid models** (i.e., meta-schema[2] compliant schemas). In this way, our approach uses a set of production rules to generate such translations. The main contributions of this paper are the following:

- An approach that uses formalized meta-models and a set of production rules to translate disparate data sources to RDFS.
- As representative cases of Big Data sources, we consider our method for JSON, XML and relational databases (RDB). In this paper, we exemplify our approach with JSON.
- A prototype tool, named ARDI (**A**utomatic Generation of **R**DFS Models from Heterogeneous Sources for **D**ata **I**ntegration), that generates an equivalent RDFS model from a given JSON, XML or RDB input.

The rest of this paper is structured as follows. Section II discusses related work. Section III illustrates how a meta-modeling approach can be used for translating disparate data sources to the canonical model. In Section IV, we provide a proof of concept using a prototype and outline its implementation. Section V concludes the paper.

II. Related Work

Ontology-based data integration systems are currently a trend for Big Data [6], [25]. However, these systems require huge manual efforts to generate the required constructs (i.e., mappings and target ontology). As a consequence, these systems are barely used by organizations due to their inherent complexities and required expertise. To the best of our knowledge, [31] and [17] are the only attempts to automate the creation of such constructs in the Big Data field. However, the approach in [31] presents a methodology, following a traditional requirement elicitation process based on the sources at hand and the approach that is employed in [17] is limited to RDB. As a consequence, the amount of manual work is still huge. For this reason, there are several works claiming for new solutions supporting the automation of data integration [15], [30].

Nevertheless, there are some efforts in ontology construction, mainly in the SW field, that can help in producing such constructs and relate to our work. In short, one may develop ontologies (and the associated mappings) manually or process existing data sources to generate them automatically. When it comes to data integration, the former is *time consuming* task that *renders unfeasible* for large systems [35] whereas the latter is a low-cost and efficient method.

In the literature, a number of researchers have recognized the importance of SW data modeling frameworks and have proposed different approaches for the migration of (semi-)structured data sources to the SW. These approaches can generally be classified as instance-level [2], [9], [20], [29] and schema-level [3], [12], [23], [34] translations. Instance-level approaches aim at generating a semantic representation of the data (instances) without generating schema information. Several frameworks (D2RQ [4], R2RML [8], RML [9], X3ML [21]) exist to let end-users express rules to translate instances from any kind of source to SW models. However, it is up to the system designer to come up with such rules. In this regard, JSON-LD [32] is a viable option for the direct translation of JSON documents to the SW.

Opposite to instance-level approaches, schema-level approaches do not translate data but just schema information (e.g., relational schema to OWL [23], UML class diagrams to OWL [12], UML class diagrams to RDFS [34]). In [23], the authors introduced philosophical principles that directly map RDB schema elements into OWL TBOX representations. They have also used these principles to evaluate the performance of existing tools for the automatic conversion of RDB to ontologies. [12] proposed a mathematical framework based on schema mappings to convert UML class diagrams to OWL. In this study, a model-to-text transformation was used to generate the textual representation of class diagrams from the source schema. Then, they proposed mapping rules between UML class diagram elements and OWL constructs. They also claim that the source schema representations and the target schema were validated, but no validation proof is provided. In [34] authors introduced a method to produce RDF schema from UML class diagrams. They also introduced a tool for implementing their approach. However, their input file needs to be XML-coded, which hinders the flexibility of their approach.

All in all, although the aforementioned approaches contribute to translating schema information to SW models, they

[2] In this paper we use the terms meta-schema and meta-model interchangeably.

(i) do not guarantee to produce sound meta-model compliant schemas, (ii) do not fully cover all schema elements that we may find in semi-structured data models (e.g., arrays in JSON), and (iii) ignore the RDFS meta-model. As a conclusion, the models generated lack a common understanding, since models are generated without fixed semantics. Thus, we propose a solution to these problems based on a meta-modeling approach.

III. Approach

Modeling languages have played an essential role in organizing information in the SW. RDFS is a primitive ontology language that allows meta-modeling. In this section, we explain how to generate RDFS representations from different input data sources.

In order to extract an RDFS representation from an input data source, we rely on two main processes, namely extract *the schema (potential schemas of sample source files) and translate them to RDFS. The extract source schema process* is responsible for the generation of a schema from an input data source that conforms to the source meta-model. The translation process uses a set of pre-defined production rules (that must be defined beforehand) to automatically translate the extracted source schemas to an equivalent RDFS model based on both the source and RDFS meta-model constructs. The overall process is depicted in Fig. 1 (presented as a BPMN diagram to show the flow of activities) and the detail of each process is discussed in the following subsections.

When presenting our method in the subsequent sections, we use JSON as an exemplary use case. Specifically, JSON documents from the Barcelona city hall providing information about public transports in the city (see a fragment in Fig. 2). We have also demonstrated our approach for XML and RDB [3]

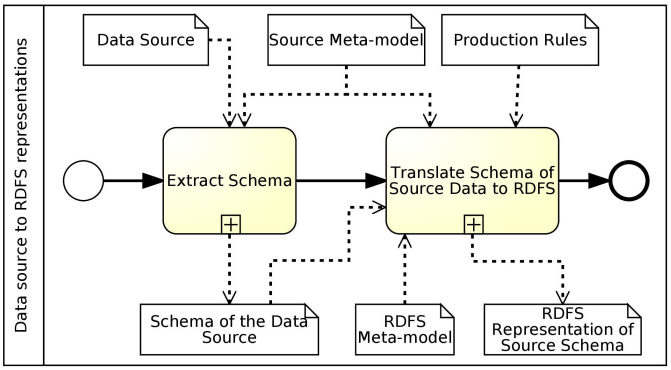

Fig. 1: Overall process for translating data sources to RDFS


```
{
  "id":1,
  "type":"BIKE",
  "address":{
  "streetName":"Gran Via Corts Catalanes",
  "streetNumber":760},
  "coordinates":[41.397952,2.180042],
  "nearbyStations":[{
  "id":24,
  "type":"Metro",
  "distance":500},{
  "id":426,
  "type":"Bus",
  "distance":367}]
}
```


Fig. 2: Running example (excerpt of the stations.json file)

A. Extraction of schemas from data sources

Semi-structured data sources are known to be schemaless (i.e., without a fixed schema for all instances). However, there is always an implicit schema. At this stage, our approach elicits the implicit schema[4]. This process takes sample data source files and the source meta-model as input and produces a meta-model compliant data source schema. Our approach relies on linguistic meta-modeling to ensure the correctness of the resulting schema. Accordingly, it reads the data source files and parses them according to the meta-schema. Thus, the source meta-model is used to constrain the extracted schema elements.

First, if not done yet, for each data model we must create a representation of its meta-model in First Order Logic (FOL). For our running example, we generated the JSON meta-schema depicted in Fig. 3 (represented as a UML class diagram to facilitate its comprehension). There, a JSON document is represented as a *Document*, JSON object as *ObjectClass*, JSON pairs (key/value) as *Attribute* and *ValueType*. Simple value types such as string, boolean and date are represented as *Primitive*, and complex values as *Array* or *Reference*. The JSON schema meta-model describes the basic constructs and the relationships between them. As can be seen, a document consists of an object class, and an object class may have a set of attributes. Each attribute has a string name and a value type which can be a primitive, an array or a reference to an object class. An array can consist of either a primitive type, a reference or another array. Finally, a reference type is composed of an object class.

We next formalize, in FOL, the general knowledge presented in the meta-model. More specifically, we are interested in expressing the meta-model *facts* (i.e., entities and relationship types). In FOL, we represent entities by constants, facts as ground formulas (i.e., formulas without variables), and predicates are used to denote properties of entities and relationships among them. Predicates have an associated arity (i.e., number of arguments), which might be one or some other finite value. An abstract description that represents entities, relationships among them and categorization between them is

[3] In case of interest, more detailed information can be found on: http://www.essi.upc.edu/dtim/ardi

[4] Note that for structured data sources, such as RDB, this step can be skipped.

called *information base* [26]. For example, for the running example, the *information base* contains:

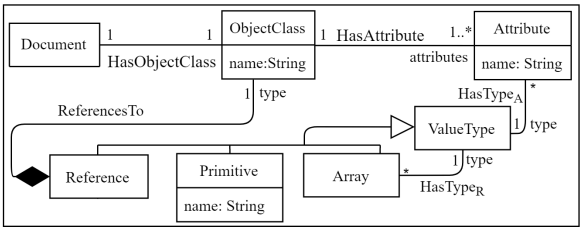

Fig. 3: JSON Schema Meta-model (inspired in [16])

- *ObjectClass(o)* representing that the constant *o* is an *ObjectClass*;
- *Attribute(a)* representing that the constant *a* is an *Attribute*;
- *ObjectClassName(o, stations)* representing that the object class *o* has name *stations*;
- *AttributeName(a,streetName)* representing that the attribute *a* has the name *streetName*;
- *HasAttribute(o,a)* representing that the object class *o* has the attribute *a*. That is, the object class with the name *stations* has the attribute with the name *streetName*.

The extracted schema does not only need to conform to the source meta-model but also satisfy its constraints. Thus, the JSON meta-schema defines some constraints that need to be satisfied by JSON schemas. Specifically, these constraints are constraints on generalizations, referential integrity constraints, cardinality constraints, and key constraints. Constraints on generalizations define the *IsA* relationships, the disjointness and covering constraints in the JSON meta-schema. Referential integrity constraints guarantee that each participant entity of a relationship is an instance of its corresponding type. The formalization of these constraints in FOL, including cardinality and key constraints, is shown in Table I.

Once the meta-schema has been formalized in FOL (this is a manual task to be done once per data model), we parse representative instances according to the meta-model in order to extract the implicit schema. For example, consider the generation of a JSON schema shown in Fig. 2. By means of its syntactic rules, we would identify the document, object classes, attributes, etc.

Note that this parsing process may be triggered once for several sample documents. For example, if a MongoDB collection stores 5 different types of JSON documents, the process must be triggered for each document and, as a result, we will generate 5 different meta-model compliant schemas describing each of the input documents. The integration of these schemas into one should be done in subsequent steps not covered in this paper. In the next section we refer to the process triggered for each generated schema.

B. Translation from extracted source schema to RDFS

The next step is to translate the schema of the input data source to RDFS. For each output of the previous process, we execute this step once. We require the RDFS meta-model as input (see Fig. 4)[5]. Also, this process relies on a set of production rules that define the translation from the schema of the input data source to equivalent RDFS representation.

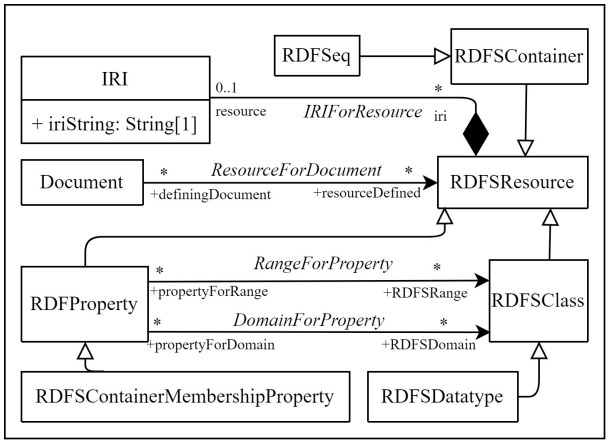

Fig. 4: RDF Schema Meta-model (extracted from [28])

Basically, the translation step is the application of a set of rules. Using a production rule, we can map the source meta-model constructs to the target (i.e., RDFS) meta-model constructs. Each production rule is represented as a logical axiom with left-hand side (LHS) and right-hand side (RHS), denoted by $LHS \Rightarrow RHS$. Thus, if LHS holds RHS must hold too. Each logical axiom consists of either unary or binary predicates representing the knowledge from source and target meta-models. For example, the following is a production rule that translates each instance of JSON *ObjectClass* to an *RDFSClass* instance with a new IRI derived from the *ObjectClassName*:

$$ObjectClassName(o, oName) \Rightarrow RDFSClass(c) \wedge IRIForResources(c, i) \wedge iriString(i, f_{class}(oName))$$

where: *o* is an instance of *ObjectClass* and *oName* is a string that identifies *o*. $f_{class}(oName)$ is a function that generates an IRI from the string *oName*.

Importantly, given an *oName* there is only one *RDFSClass* with such IRI. Also, our production rules use the information from both the source meta-model (e.g., JSON meta-model) and the RDFS meta-model to guarantee that any valid (e.g., JSON) model might be processed. Aligning the schema elements with the meta-model constructs offers the following advantages:

- Conforming the source schema elements with the source schema meta-model constructs enables us to preserve the behavior of the source schema in the translation process.
- Further, conforming the translated elements with the RDFS meta-model constructs will result in the RDFS representations to be conformant with the RDFS meta-model.
- Consequently, with respect to the advantages mentioned above, the former aims at guaranteeing that our pro-

[5]The FOL representation can be found at: http://www.essi.upc.edu/dtim/ardi

TABLE I: Logical representation of JSON meta-model constraints

Types of Constraint	*Logical Representation*
Constraints on generalizations	$\forall v(ValueType(v) \Rightarrow Reference(v) \vee Primitive(v) \vee Array(v))$ $\forall r(Reference(r) \Rightarrow ValueType(r))$ $\forall p(Primitive(p) \Rightarrow ValueType(p))$ $\forall y(Array(y) \Rightarrow ValueType(y))$ $\forall r(Reference(r) \Rightarrow \neg Primitive(r))$ $\forall r(Reference(r) \Rightarrow \neg Array(r))$ $\forall p(Primitive(p) \Rightarrow \neg Array(p))$
Referential integrity constraints	$\forall d, o(HasObjectClass(d, o) \Rightarrow Document(d) \wedge ObjectClass(o))$ $\forall o, a(HasAttribute(o, a) \Rightarrow ObjectClass(o) \wedge Attribute(a))$ $\forall a, v(HasType_A(a, v) \Rightarrow Attribute(a) \wedge ValueType(v))$ $\forall r, o(ReferenceTo(r, o) \Rightarrow Reference(r) \wedge ObjectClass(o))$ $\forall y, v(HasType_R(y, v) \Rightarrow Array(y) \wedge ValueType(v))$ $\forall o, m(ObjectClassName(o, m) \Rightarrow ObjectClass(o) \wedge String(m))$ $\forall a, m(AttributeName(a, m) \Rightarrow Attribute(a) \wedge String(m))$ $\forall p, m(PrimitiveName(p, m) \Rightarrow Primitive(p) \wedge String(m))$
Cardinality constraints	$\forall d(Document(d) \Rightarrow \exists! o(HasObjectClass(d, o)))$[a] $\forall o(ObjectClass(o) \Rightarrow \exists^{\geq 1} a(HasAttribute(o, a)))$ $\forall a(Attribute(a) \Rightarrow \exists! v(HasType_A(a, v)))$ $\forall r(Reference(r) \Rightarrow \exists! o(ReferencesTo(r, o)))$ $\forall y(Array(y) \Rightarrow \exists! v(HasType_R(y, v)))$ $\forall o(ObjectClass(o) \Rightarrow \exists! m(ObjectClassName(o, m)))$ $\forall a(Attribute(a) \Rightarrow \exists! m(AttributeName(a, m)))$ $\forall p(Primitive(p) \Rightarrow \exists! m(PrimitiveName(p, m)))$
Key constraints	$\forall o_1, o_2(ObjectClassName(o_1, o_1Name) \wedge ObjectClassName(o_2, o_2Name) \wedge (o_1 \neq o_2) \Rightarrow (o_1Name \neq o_2Name))$ $\forall o, a_1, a_2(HasAttribute(o, a_1) \wedge HasAttribute(o, a_2) \wedge AttributeName(a_1, a_1Name) \wedge AttributeName(a, a_2Name) \Rightarrow (a_1Name \neq a_2Name))$ $\forall v(HasType_A(a_1, v) \wedge HasType_A(a_2, v) \Rightarrow (a_1 \neq a_2))$ $\forall p_1, p_2(PrimitiveName(p_1, p_1Name) \wedge PrimitiveName(p_2, p_2Name) \Rightarrow (p_1Name \neq p_2Name))$ $\forall o(HasAttribute(o, a_1) \wedge HasAttribute(o, a_2) \Rightarrow (a_1 \neq a_2))$ $\forall v(HasType_R(y_1, v) \wedge HasType_R(y_2, v) \Rightarrow (y_1 \neq y_2))$

[a]Note that $\exists! y P(x, y)$ mean there is a unique y such that $P(x, y)$ is true.

duction rules are semantically correct whereas the latter refers to their syntactic correctness.

Finally, SW resources must always be identified by an IRI. We provide a mechanism to automatically generate IRIs for elements of the target semantic model. Thus, we define auxiliary functions to be used by the production rules to automatically generate IRIs for all the resources of the target schema. Note that these functions can be reused for any other data model talking in terms of objects, attributes, data types and arrays (simply adapting them to the new meta-model).

Let us consider now our running example. For JSON, we have defined 8 production rules. In this case, we only have one strong element (i.e., object class) and R_1 translates instances of this element to an RDFS class. All the other rules translate weak elements that require to translate other meta-model elements first. We exemplify the the production rules for JSON with R_1, R_2 and R_3.

Rule 1: JSON ObjectClasses are translated to RDFS Classes We create an *RDFSClass* to represent each JSON *ObjectClass* instance. This rule gets all instances of *ObjectClass* from the JSON meta-model using the predicate *ObjectClassName*, and then it generates an instance of *RDFSClass* with an IRI derived from the name of the object classes. Fig. 5 shows an example of such translation for the object class with name *stations* (Fig. 2). Note that sc : is a prefix for $www.BDIOntology.com/schema/$

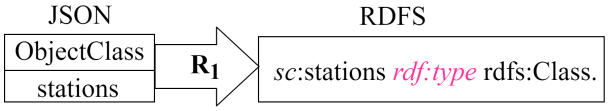

Fig. 5: An example of RDFS class generation

Rule 2: JSON Attributes are translated to RDF Properties R_2 is a rule for a weak element (attributes), and therefore it requires instances of JSON *ObjectClasses* to have already been translated to RDFS (i.e., R_1 is triggered first). For the running example (Fig. 2), an instance of RDFProperty that can be identified by an IRI generated from the object class name (e.g., stations) and attribute name (e.g.,id, type) using the function f_{att}[6] (i.e., $f_{att}(stations, id)$, $f_{att}(stations, type)$). The domain of this property is an instance of RDFS class identified by an IRI obtained using the function $f_{class}(stations)$.

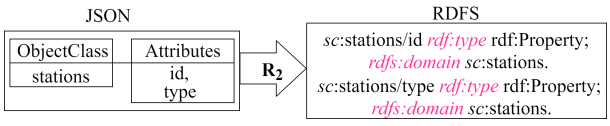

Fig. 6: An example of JSON Attribute to RDFProperty translation

[6]f_{att} is a function that returns an IRI from the strings objectName and attributeName (i.e., $f_{att}(oName, aName)$)

Rule 3: Range representation for Properties with Primitive Type

Besides a domain, an instance of *RDFProperty* includes a range that states the classes of its values. The range of properties is obtained from the JSON attributes value types. Since an attribute in JSON can have primitive, array or reference value types, the transformation from value types to range of properties is handled by different rules. Rule R_3 is used to translate the primitive value type of an attribute to an instance of *RDFDatatype*. For example, Fig. 7 shows the xsd:string *RDFDatatype* as the range of the property with the IRI calculated from $f_{att}(stations, type)$.

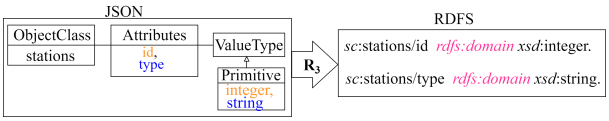

Fig. 7: An example of attribute value to property range mapping

Rules R_4, R_5, R_6, R_7 and R_8 have been defined for mapping arrays and reference types to the RDFS constructs. More specifically, R_4 translates references to RDFS class that represents the range for properties and R_5 translates JSON arrays to an instance of both an RDFS class and RDF sequence. While an array is translated to an instance of both RDFS class and RDFS sequence, we define an instance of container membership property whose domain is the RDFS class and RDFS sequence created using R_5, and its range corresponds to the element of the given array. To this end, rules R_6, R_7 and R_8 addresses primitive, reference and array of array elements, respectively. Detailed information can be found at http://www.essi.upc.edu/dtim/ardi.

IV. Prototype

We created a tool named ARDI[7] to implement our approach. It follows the service-oriented architecture and implemented in Java by making use of Jena [22] to translate source schemas to RDFS triples. It accepts JSON, XML and RDB as input, extracts the input data source schema and translates it to RDFS using the previously presented translation rules. For RDB, it connects to an existing database, reads the database schema and translates it to RDFS.

We tested ARDI using Open Data repositories such as the Barcelona city repository on public transport (see Section III). Moreover, to further validate our production rules by means of external tools we chose Protégé [24] to make sure if the generated RDFS representation was interpreted by it. All the RDFS models generated by ARDI were successfully interpreted and visualised by Protégé. For example, Fig. 8 shows the graphical representation of the RDFS schema corresponding to the excerpt in Fig. 2.

[7]Available at: http://dtim.essi.upc.edu/shumet/ardi

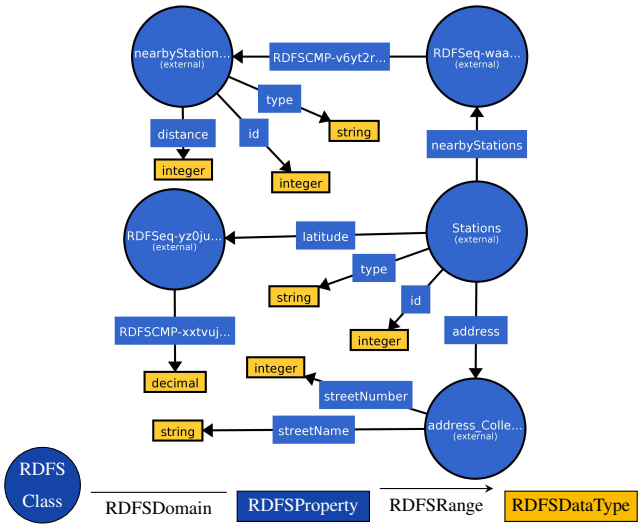

Fig. 8: An example of RDFS graph

V. Conclusion

In this paper, we discuss the need of a meta-modeling approach for the automatic translation of heterogeneous data sources to semantic models. We specify the translation at the meta-model level using a set of production rules between the source and RDFS meta-models. Our approach is generic as to consider any data source, and we explained how to formalize the source data model and apply production rules to guarantee a valid result. We have exemplified this process with JSON and implemented a tool that does so for JSON, XML, and RDB.

Altogether, this is an important step towards Big Data Integration. As identified in [13] there is a lack of tools to automatically extract and integrate data from heterogeneous sources. Our approach automates the first step by extracting a common representation of the sources on a common canonical data model. To the best of our knowledge, no other similar approach/tool exists for a typically manual and error-prone task. The automatic integration of heterogeneous data sources requires several efforts. For the future, we plan to support the next stages in data integration and develop an end-to-end semi-automatic integration framework building on top of the results generated by the approach presented in this paper.

Acknowledgments

This work has been partly supported by the GENESIS project, funded by the Spanish Ministerio de Ciencia e Innovacin under project TIN2016-79269-R, and by the European Commission through the Erasmus Mundus Joint Doctorate Information Technologies for Business Intelligence-Doctoral College (IT4BI-DC).

References

[1] Acharjya, D.P., Ahmed, K.: A survey on big data analytics: challenges, open research issues and tools. Int. J. Adv. Comput. Sci. Appl **7**(2), 1–11 (2016)

[2] Arenas, M., Bertails, A., Prudhommeaux, E., Sequeda, J.: A direct mapping of relational data to rdf. W3C recommendation **27** (2012)

[3] Bakkas, J., Bahaj, M., Marzouk, A.: Direct migration method of rdb to ontology while keeping semantics. IJCA **65**(3) (2013)

[4] Bizer, C., Seaborne, A.: D2rq-treating non-rdf databases as virtual rdf graphs. In: Proceedings of the 3rd ISWC, 2004. vol. 2004. Proceedings of ISWC2004 (2004)

[5] Botoeva, E., Calvanese, D., Cogrel, B., Corman, J., Xiao, G.: A generalized framework for ontology-based data access. In: AI*IA. pp. 166–180. Springer (2018)

[6] Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering sparql queries over relational databases. Semantic Web **8**(3), 471–487 (2017)

[7] Chang, D.T., Kendall, E.: Metamodels for rdf schema and owl. In: MDSW 2004, Monterey, USA (2004)

[8] Das, S., Sundara, S., Cyganiak, R.: R2rml: Rdb to rdf mapping language.(2012) https://www. w3. org. Tech. rep., TR/r2rml

[9] Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: Rml: A generic language for integrated rdf mappings of heterogeneous data. In: LDOW (2014)

[10] Dou, D., Wang, H., Liu, H.: Semantic data mining: A survey of ontology-based approaches. In: ICSC. pp. 244–251. IEEE (2015)

[11] Gašević, D., Djurić, D., Devedžić, V.: Bridging mda and owl ontologies. Journal of Web Engineering **4**(2), 119–134 (2005)

[12] Gherabi, N., Bahaj, M.: A new method for mapping uml class into owl ontology. SEDEXS pp. 5–9 (2012)

[13] Golshan, B., Halevy, A., Mihaila, G., Tan, W.C.: Data integration: After the teenage years. In: SIGMOD-SIGACT-SIGAI. pp. 101–106. ACM (2017)

[14] Halevy, A., Doan, A., Ives, Z.: Principles of data integration (2012)

[15] Halevy, A.Y.: Technical perspective: Building knowledge bases from messy data. Commun. ACM **60**(5), 92 (2017). https://doi.org/10.1145/3060584, https://doi.org/10.1145/3060584

[16] Izquierdo, J.L.C., Cabot, J.: Discovering implicit schemas in json data. In: ICWE. pp. 68–83. Springer (2013)

[17] Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I., Pinkel, C., Skjæveland, M.G., Thorstensen, E., Mora, J.: Bootox: Practical mapping of rdbs to owl 2. In: ISWC. pp. 113–132. Springer (2015)

[18] Kobryn, C.: Uml 2001: a standardization odyssey. Communications of the ACM **42**(10), 29–37 (1999)

[19] Laney, D.: 3d data management: Controlling data volume, velocity and variety. META group research note **6**(70), 1 (2001)

[20] Lefrançois, M., Zimmermann, A., Bakerally, N.: A sparql extension for generating rdf from heterogeneous formats. In: ESWC. pp. 35–50. Springer (2017)

[21] Marketakis, Y., Minadakis, N., Kondylakis, H., Konsolaki, K., Samaritakis, G., Theodoridou, M., Flouris, G., Doerr, M.: X3ml mapping framework for information integration in cultural heritage and beyond. IJDLS **18**(4), 301–319 (2017)

[22] McBride, B.: Jena: A semantic web toolkit. IEEE Internet computing **6**(6), 55–59 (2002)

[23] Mogotlane, K.D., Fonou-Dombeu, J.V.: Automatic conversion of relational databases into ontologies: A comparative analysis of protégé plug-ins performances. arXiv preprint arXiv:1611.02816 (2016)

[24] Musen, M.A.: The protégé project: a look back and a look forward. AI matters **1**(4), 4–12 (2015)

[25] Nadal, S., Romero, O., Abelló, A., Vassiliadis, P., Vansummeren, S.: An integration-oriented ontology to govern evolution in big data ecosystems. Information Systems **79**, 3–19 (2019)

[26] Olivé, A.: Conceptual modeling of information systems. Springer Science & Business Media (2007)

[27] OMG: Ontology Definition Metamodel. OMG Book (September), 338

[28] OMG: Ontology Definition Metamodel. OMG Book (September), 338 (2014), http://www-inf.it-sudparis.eu/SIMBAD/tools/SoSeC-BPO/formal-14-09-02.pdf

[29] Santipantakis, G.M., Kotis, K.I., Vouros, G.A., Doulkeridis, C.: Rdf-gen: Generating rdf from streaming and archival data. In: WIMS. p. 28. ACM (2018)

[30] Sarma, A.D., Dong, X.L., Halevy, A.Y.: Uncertainty in data integration and dataspace support platforms. In: Schema Matching and Mapping, pp. 75–108. Springer (2011)

[31] Sequeda, J.F., Miranker, D.P.: A pay-as-you-go methodology for ontology-based data access. IEEE Internet Computing **21**(2), 92–96 (2017)

[32] Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Lindström, N.: Json-ld 1.0. W3C Recommendation **16**, 41 (2014)

[33] Stonebraker, M., Ilyas, I.F.: Data integration: The current status and the way forward. IEEE Data Eng. Bull. **41**(2), 3–9 (2018)

[34] Tong, Q., Zhang, F., Cheng, J.: Construction of rdf (s) from uml class diagrams. Journal of computing and information technology **22**(4), 237–250 (2014)

[35] Xu, Z., Ni, Y., He, W., Lin, L., Yan, Q.: Automatic extraction of owl ontologies from uml class diagrams: a semantics-preserving approach. World Wide Web **15**(5-6), 517–545 (2012)