Generation of Web pages for Public Scientific Databases Using Schema.org

Josef Hardi¹, Kody Moodley², John Graybeal¹, Michel Dumontier², Mark A. Musen¹

¹Stanford Center for Biomedical Informatics Research, Stanford University, USA
{johardi, jgraybeal, musen}@stanford.edu
²Institute of Data Science, Maastricht University, The Netherland
{kody.moodley, michel.dumontier}@maastrichtuniversity.nl

Abstract. While much effort in applying schema.org concentrates on popular text, such as news articles, blogs, or restaurant reviews, the scientific data on the Web have received less attention. For example, pages from public database websites (e.g., DrugBank, PubMed, NASA, NOAA) are almost never presented using schema.org. This absence prevents Web search engines from applying more advanced search features, such as filtering or ambiguity resolution, to information generated by these websites. In this paper, we describe software for generating schema.org-compliant Web pages out of raw metadata used to describe scientific registries or datasets using an extract-transform-load (ETL) pipeline. With this software, data elements that can be mapped to schema.org are automatically extracted and transformed into JSON-LD and then loaded into HTML source. We also present a declarative mapping language to facilitate the data mapping in the extraction process. The result is a framework that public databases can use to publish Web pages that are semantically indexable by search engines. We show that annotating scientific data using schema.org can be done effectively using a well-defined data mapping and ETL processes.

Keywords: Semantic content authoring, schema.org, linked data, web technology

1 Introduction

The term schema.org has become widespread among webmasters in recent years, reflecting the popularity of incorporating structured knowledge into HTML Web pages. Schema.org has emerged as the de facto standard for publishing semantically described structured data on the Web. However, this practice is largely for popular Web content such as news articles, blog posts, or reviews, and it is less common for Web pages from scientific domains. A survey from WebDataCommons.org [1] in 2017 shows that few websites annotate their pages using scientific-related terms from schema.org, such as MedicalEntity, Drug, ScholarlyArticle or Dataset, although there is an upward trend since 2013. Such absence will prevent Web search engines from applying more advanced search features, such as filtering [2] or ambiguity resolution [3], to search results that come from these websites.
We defined public scientific data as all kind of data registries and datasets about scientific activities that are made available in public database websites for anyone to analyze and reuse. Typically they are government-generated data or made by academic communities. DrugBank\(^1\), PubMed\(^2\), NASA Data Portal\(^3\) and NOAA Data Portal\(^4\) are some examples of these public database websites and they share several similarities about their data: 1) these are public data and are thus easy to obtain and redistribute, 2) they tend to include well-defined metadata and 3) the whole distribution is trustworthy and often high-quality. These positive characteristics are indeed the supporting factors that contribute to the development work carried out in this work.

We explore the use of data mapping and extract-transform-load (ETL) processes to produce schema.org-compliant Web pages out of the raw metadata used to describe scientific registries and datasets. The goal is to make the web pages semantically indexable by search engines so that the data stored in those databases might better reach the Internet users through presentation in search results. To our knowledge, we are the first to implement a tool that augments scientific data with schema.org in well-formed HTML pages.

The remainder of this paper is organized as follows: Section 2 reviews some tools used in semantic content authoring and considers how their performance is not likely to be suitable for data in large quantities. Section 3 describes our proposed solution in detail. Section 4 shows the demo application in action and lastly, Section 5 gives an outlook on future work and concludes the paper.

2 Related Work

The introduction of microformats, such as hCard, hCalendar and hAtom, predates the practice of embedding semantic meaning into Web content. The term semantic content authoring\(^4\) emerged as a process to compose textual information together with a semantic representation, often with the help of a semi-automatic tool.

Some approaches and tools exist for semantic content authoring for schema.org. One of them is RDFaCE\(^5\), which is a plugin for the rich text editor in a content management system (CMS) that augments text editing with interactive concept tagging. Google Data Highlighter\(^6\) is another interactive tool that works directly with the text on a Web page. In Google Data Highlighter, users select sections of data in the text with a mouse, and Google will automatically learn the data locations right from the page and generate a markup template. The next time Google crawls the site, it will use the markup information in the template to generate the structured data. Another tool from Google called the Structured Data Markup Helper\(^7\) is similar to the data highlighter, but a key difference is that users must embed the output markup into the HTML source by themselves.

---

1 http://drugbank.ca
3 https://data.nasa.gov
4 https://data.noaa.gov
These tools, however, have restricted applicability in working with a large data stream. RDFaCE uses a front-end user interface to obtain the semantic tagging from users which must be done manually by hand. The tool does not offer an internal parser to check the document structure and make automatic guesses about the content. A similar situation is also true for the Google Structured Data Markup Helper tool, which requires manual hand patching of the Web pages. By contrast, the Google Data Highlighter tool has the ability to automatically process multiple pages as long as they are displaying the data consistently in the site, a strategy which is often found in public database websites. Although it still requires human assistance to establish the markup template, the highly reusable template makes the approach scalable. Unfortunately, at the time of writing, the tool had a very limited number of supported schema.org types, and thus had limited applicability for other types of information, including scientific-related content. Besides this limitation, we see great potential for the data highlighter tool in meeting the demands of scientific data annotation on the Web.

In our research group at Stanford, we are developing a semantic metadata authoring tool called the CEDAR Workbench [8] that can facilitate metadata creation using schema.org vocabulary in a form-filling interface. The work presented in this paper will contribute to the extension of the workbench to support public metadata publication.

3 Method

The aim of our work is to provide an implementation of a software tool that helps public database owners generate schema.org markup from their data sources automatically. We will show that the method is meant to be generic in the sense that it is applicable to any data store and the only constraint is that the data is required to have some structure, or a model that supports some structure.

We first illustrate the steps used in typical ETL processing to transform data from one model to another. We then show our pipeline setup used to accomplish the ETL steps and describe the mapping language used in our pipeline. This section ends with a high level description of the software implementation.

3.1 ETL Process Scenario

The term ETL comes from data warehousing and stands for extract, transform and load. The extraction process is the first step of ETL; it retrieves a desired subset of data from a source database. The acquired data is then processed further in the transformation step to make all the values and structure conform to the target schema. And finally, in the loading step, the transformed data is integrated into a new target destination. With these processes in mind, we built our ETL process scenario for the schema.org data transformation, as follows:

- Since our data extraction will involve a model translation from a source schema to the schema.org schema, the first step is to find a semantic correspondence between
the two schemas through a schema matching process [9]. Some matching evaluation techniques can be used, for example, matching through name similarity, description similarity, type similarity or value pattern and ranges. Using these techniques we will produce a mapping description that identifies the corresponding data fields in the two schemas. This step is repeated for all the data fields in the source schema, resulting in a schema map document. Involving a domain expert in the matching process will help to suggest early corrections and produce an optimal result.

• After a schema map is created, the next step consists of writing the logical data map (or shortly data map) in order to prepare the extraction process. A data map contains the actual physical data location (e.g., column names, data paths, tree nodes) that replaces the source fields in the schema map. We can translate this information to build a query for the data extraction. Once the data is extracted, the next step is to convert it into JSON-LD, which is the format we have chosen to encode the schema.org markup data for practical reasons. This step can be done using a program script that will parse the content of the extracted data and generate a JSON-LD version.

• The final step is to load the schema.org data into an HTML page. A program script is responsible for generating an HTML page while another script will inject the JSON-LD markup into the HTML head section.

In the next subsection, we will present our pipeline setup and describe the different pipe programs that carry out the above steps.

3.2 Pipeline Setup

We framed the ETL steps as a pipeline that performs a series of transformations from the source data stream to the semantic authoring pipe, to the markup exporting pipe, to the content publishing pipe and finally ends in the output stream (see Figure 1).

![Figure 1. Schema.org data markup pipeline](image)

• The semantic authoring pipe contains a program to extract the data from the source using the data map document. At this point, the field names of the extracted data will already be expressed in schema.org as set out by the mapping definitions. We will further explain the mapping language used in the mapping definition in the following sub-section.

• The markup exporting pipe contains a program to rewrite the extracted data by transforming the original structure into the JSON-LD format.
• The content publishing pipe contains a program to generate the HTML page and load the JSON-LD by inserting it into the HTML source.

As described, a pipeline allows simple operations to be combined to perform a complex task while avoiding the need for intermediate results to be saved to disk thus making it a performant strategy. A pipeline is also very configurable, allowing the pipeline designer to add or replace pipes to suit different applications.

3.3 Mapping Language

A challenge for this task was defining the mappings in a way that was unambiguous, minimalist in its representation, general-purpose, and lacking any dependency on target model languages. Since the mapping languages that we reviewed, such as RML\(^5\) or R2RML\(^6\), did not support all of these features, we developed a mapping language called CAML\(^7\) (not to be confused with the markup language developed by Microsoft [10]) to help the end users easily writing the data map.

A data map in CAML is composed of one or more mapping definitions. A mapping definition is written as a key-value pair separated by a colon and at least one space, where the key is the schema.org keyword and the value is either a data path, a data object or a constant value.

• A data path represents the physical data location in a tree hierarchy representation. The notation always starts with a slash “/” followed by a node name. The slash character is also used as a delimiter to separate multiple node names used in the path.

• A data object is a group of mapping definitions at the same indentation level. For schema.org-compatibility, every data object must have a type definition, indicated by the keyword @type and followed by a target schema.org type name. For example, the expression @type: ‘MedicalTrial’ declares the type of the data object is a medical trial.

• A constant value is any other text that is enclosed by single-quotation marks. A backslash should be used as an escape character in the text.

The syntax also supports an array value by creating multiple mapping definitions with the same key name. There is another syntax rule in CAML for expressing a nested object map, which is that it must begin with the root path description before adding the data object below it. Code Listing 1 shows a full-length example of a data map in CAML.

---

\(^5\) http://rml.io
\(^6\) https://www.w3.org/TR/r2rml/
\(^7\) CAML is a recursive acronym for “CAML is Another Mapping Language”
We used a declarative approach in designing the language to achieve simplicity while still retaining enough expressive power to define the commonly used data map expressions. The approach is also useful for decoupling conceptual modeling from implementation details, providing an advantage for future adaptations to different data extraction methods.

The RDF Mapping Language (RML) can be used instead of CAML to create the mapping definitions. In comparison to RML, CAML is an experimental mapping language that is not yet supported by an active community. The syntax notation does not support functions, such as substring or string concatenation, in its current version. However, unlike RML, which is intended to map data from various sources to the RDF data model, CAML is a generic language for simply defining mapping definitions without ties to any data model.

### 3.4 Implementation Details

The code for the pipeline and the mapping language is written in Java 8 and is available\(^8\) as an open-source software library under FreeBSD license. The library has one important interface called `Pipe` that can be chained to construct a complete pipeline. The interface has only one method, `run`, which is invoked when the output from the previous pipe arrives to the current pipe as its input. In Java 8, an interface that contains only one method is called a `functional interface` and it can be assigned to a `lambda expression` to give the code a more compact look (see Code Listing 2).

\(^8\) [https://github.com/metadatacenter/schemaorg-pipeline](https://github.com/metadatacenter/schemaorg-pipeline)
We implemented two executable programs\(^9\) using the software library, namely \textit{rdf-pipeline} and \textit{xml-pipeline}. The program \textit{rdf-pipeline} works for any triplestore with a public SPARQL endpoint. First, the program receives the endpoint address as the input parameter and opens the connection to the targeted triplestore. Once the connection is established, the program automatically generates a SPARQL CONSTRUCT\(^10\) query from the input data map and sends the query to be evaluated. Second, a series of pipes then processes the RDF output and loads it into an HTML page as JSON-LD data. We utilized the RDF4J\(^11\) library to handle the query evaluation and the data transformation in the pipeline.

The other program \textit{xml-pipeline} works for XML documents. At the start of the program, the data map is translated into an XSLT stylesheet to transform the original input XML into a \textit{schema.org} output XML. Similar to the aforementioned processes, a series of pipes then processes the output XML and loads it into an HTML page as JSON-LD data. The Java XML API and JSON-java\(^12\) library provide the XSLT processing and the data transformation in the pipeline.

### 3.5 Evaluation Result

We conducted an evaluation of the pipeline programs by assessing how well they can transform a very large dataset and how fast the program executes the task. The test was performed on an Intel 3 GHz Core i7 processor with 16 GB of RAM running macOS Sierra and Java 8.

We tested the \textit{xml-pipeline} program using data dumps from DrugBank, PubMed and ClinicalTrials. \textbf{Table 1} shows the overall test results of the program for each data source. A total of 562,430 XML documents were successfully transformed into \textit{schema.org}-compliant Web pages with an average execution time of 20.6 milliseconds per document item.

---

\(^9\) https://github.com/metadatacenter/schemaorg-pipeline-standalone

\(^10\) https://www.w3.org/TR/rdf-sparql-query/#construct

\(^11\) http://rdf4j.org

\(^12\) https://github.com/stleary/JSON-java
Table 1. Overall test results of the xml-pipeline program

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Data Volume</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrugBank</td>
<td>8,283</td>
<td>4 minutes 18 seconds</td>
</tr>
<tr>
<td>PubMed</td>
<td>299,985</td>
<td>1 hour 4 minutes 3 seconds</td>
</tr>
<tr>
<td>ClinicalTrials</td>
<td>254,162</td>
<td>1 hour 15 minutes 47 seconds</td>
</tr>
</tbody>
</table>

Previous results with the rdf-pipeline took slightly longer execution time due to the network cost, however the final test could not be performed as the Bio2RDF server was unavailable at the time of writing.

4 Demo: Schema.org Pipeline Playground

The schema.org pipeline playground is a sandbox application that helps early adopters to learn mapping with CAML and to experiment with schema.org Web page generation. Figure 3 shows the user interface of the playground which consists of three main panels (from left to right): the input data source panel, the input mapping panel and the output result panel. In the data source panel, users can either enter a SPARQL endpoint or paste XML text. In the subsequent mapping panel, users are then required to specify the data map model by specifying the mapping definitions, either in CAML or RML syntax. The output result will appear in the final panel which is a non-editable area and has three output view selections:

- The query view displays the data transformation query that is auto-generated from the input data map. If the data source is a SPARQL endpoint then it will show the SPARQL CONSTRUCT query, or if the data source is XML text then it will display an XSLT stylesheet.
- The markup view displays the JSON-LD that represents the schema.org markup data.
- The presentation view displays the HTML page source that contains both the schema.org markup data and the user-readable content. A preview option is available to display the HTML in Web browser mode.

The playground has some quick start examples to introduce basic CAML notation and to provide some real use cases from Bio2RDF\(^{13}\) and ClinicalTrials.gov\(^{14}\) for annotating such data with schema.org. The source code for the playground is also available\(^{15}\) as an open-source application under FreeBSD license. The playground is

\(^{13}\) http://bio2rdf.org

\(^{14}\) https://clinicaltrials.gov

\(^{15}\) https://github.com/metadatacenter/schemaorg-pipeline-playground
Figure 3. The playground UI showing its three panels (from left to right): the data source panel, the mapping panel, and the result panel

5 Conclusion

In this paper, we have described software to generate schema.org-compliant Web pages from structured raw data. The software is based on the implementation of an ETL process used in data warehousing and it can handle RDF and XML data in large quantities. We have tested the pipeline to generate 562,430 schema.org-compliant Web pages using data from DrugBank, PubMed and ClinicalTrials. We also have described the mapping language used to make the data mapping definitions which are later applied in the data extraction process. Finally, we have described a playground application that can help early adopters to learn the technology and develop scenarios for their needs. Altogether we have showed that annotating scientific data using schema.org can be accomplished in a simple, yet systematic way using the framework we have presented, which is in turn based on a well-defined data mapping and ETL processes. This resource is useful for data owners, and not exclusive only to scientific data owners, to help augmenting their source data with schema.org data and publishing both in a well-formed HTML page.

For the future work on the pipeline code, we intend to support more extraction data source types, such as relational databases and JSON objects. We also will publish the generated HTML pages online to allow Web search engines to index the schema.org summaries of the scientific data. Finally, we will explore the utility of these schema.org-compliant Web pages in other applications, for example to provide faceted search over that content.
Acknowledgements. This research was supported by the National Institutes of Health (NIH) under grants 5U54AI117925-04 and 5U54AI117925

References

1. Web Data Commons, http://webdatacommons.org