Atılım University
Department of Computer Engineering

Translation from XSD to JSON Schema

A short introductory report on conversion of two popular data interchange formats.

Tuğçe Kaman* Tolga Üstünkök†

January 12, 2022

CMPE541 - Advanced Databases‡

*M.Sc. candidate in Computer Engineering at Atılım University.
†Ph.D. candidate in Software Engineering at Atılım University.
‡Authors thank to Dr. Damla TOPALLI for her great support and contributions.

Contents

1 Introduction

Today, almost every web application is working asynchronously. While a long-running user task is running in the background, the web application shows a waiting animation to the user. Asynchronous web applications immensely increase user satisfaction and provide informative feedback to the user.

Getting the necessary information to start a background task and informing the user that the background process finished its job is made possible by sending and receiving messages between server and client. These messages should have well-defined structures and standards. They should also structure the transmitted data and store it when necessary. These messages can be plain text as well as proprietary binary data. Both have their advantages and disadvantages. Employing plain text messages over HTTP has its advantages against using binary data. Because of that, text-based markup languages are developed and have quickly become popular. Two of the most famous ones are Extensible Markup Language (XML) and JavaScript Object Notation (JSON). Since these languages have similar/interchangeable tasks, converting one to another can be very useful in some contexts.

1.1 Motivation

XML Schema Definition (XSD) is a very stable and widespread set of elements to be used with XML to transfer data back and forth across the server and host. W3C published its recommendation of XSD in 2001. Since then, it has been maintained and protected by various standards.

The history of JSON Schema is quite similar to the one of XSD, with one key difference. The syntax of JSON is immensely influenced by the JavaScript dictionary notation. This property is very web developer-friendly since they are often familiar with JavaScript. In addition, JSON was developed mainly considering web technologies such as HTTP, Flash, and JavaScript.

All of these notches suggest some sort of conversion should be made possible from XSD to JSON. However, the literature has little research on the topic. Therefore, a summary of the existing studies can be advantageous to have for developing further research ideas. Besides, due to the popularity of XSD, some JSON suitable data have already been stored in XSD rather than JSON. The conversion of this type of data can be beneficial. Also, transmitting messages with JSON is much more optimized in some modern environments.

1.2 Preliminaries

In general, a schema describes the characteristics of an object and its relations with other objects. A schema defines elements and attributes of elements to represent an object such as a web page. Examples of such a schema can be HTML, XML. A schema does not have to be a document. It can be a byte stream or a database record. In this report, a schema is always interpreted as a document.

To understand the actual need for conversion, one should understand the underlying formats. In this section, a summary of both XSD and JSON Schema is presented. The explanations for each format are also supported with brief examples.

1.2.1 XML Schema Definition

Fallside et al. [1] start giving an example XML document (but not a schema yet) on a home products ordering and billing application in their primer work on XML Schema. Their example document can be seen in Listing 1.

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
<shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
</shipTo>
<billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
</billTo>
<comment>Hurry, my lawn is going wild<!/comment>
<items>
    <item partNum="872-AA">
        <productName>Lawnmower</productName>
        <quantity>1</quantity>
        <USPrice>148.95</USPrice>
        <comment>Confirm this is electric</comment>
    </item>
    <item partNum="926-AA">
        <productName>Baby Monitor</productName>
```

```
        <quantity>1</quantity>
        <USPrice>39.98</USPrice>
        <shipDate>1999-05-21</shipDate>
    </item>
</items>
</purchaseOrder>
```

Listing 1: An XML document for a home products ordering and billing application.

Listing 1 starts with a version identification and continues with several subelements (elements that contain other elements) that are spanned by a top-level element called "purchaseOrder". Each element has its closing tag if they are not one-liners. Elements that carry subelements or attributes are called complex types. Other elements that only have numbers, strings, dates, etc. are called simple types.

An XML document is said to be an XML schema when its elements and subelements have "xsd" namespace associated with them. A partial example from Fallside et al. is given in Listing 2.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:annotation>
    <xsd:documentation xml:lang="en">
    Purchase order schema for Example.com.
    Copyright 2000 Example.com. All rights reserved.
    </xsd:documentation>
</xsd:annotation>

<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

<xsd:element name="comment" type="xsd:string"/>

<xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items"  type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

...

</xsd:schema>
```

Listing 2: A partial XML schema for a home products ordering and billing application.

The "xsd" namespace is not a special name and can be anything. However, to prevent each author has their own namespaces "xsd" is used by convention. It also describes the same type in different schemas, like "xsd:string". Although the originating library for a namespace can be the same, the name of the namespace should also be consistent throughout different schemas.

1.2.2 JSON Schema Definition

JSON is the most popular data exchange format to communicate with an API [6]. However, it lacks a standard to permit its users to be consistent throughout multiple projects. JSON Schema [4] is such an effort. Validators of JSON Schema cannot agree upon the syntax other than the most general parts since JSON Schema is still a work in progress.

```
{
    "type": "object",
    "properties": {
    "Country": {"type": "string"},
    "City": {"type": "string"},
    },
    "required": ["Country", "City"]
    "additionalProperties": false,
}
```

Listing 3: A JSON Schema that describes an API call structure to validate the integrity of a request.

One job of JSON Schema is to provide a framework to validate the compliance of requests and responses with the API. For example, the JSON Schema shown in Listing 3 describes an API call to a weather application.

This schema informs the validator to accept a JSON document of an object type. That is, it must consist of key-value pairs. There must be keys *Country* and *City* which must be strings and they are required. There cannot be any more keys.

2 Methods

In this section, a summary of the literature on the subject is given. First, a generic transşation process is described. Then, more specific studies are presented.

2.1 A Generic Translation Process From XSD to JSON Schema

JSON is becoming more and more popular as a data interchange format, but not as common as XML, which has recently risen in web service applications. However, converting the XML data type directly to the JSON data type results in a format error in the original data. One can make the XML-to-JSON conversion error-free by converting the XML Schema Definition (XSD) to the appropriate JSON Schema. According to the study of Guo et. al. [3], XSD is deployed to create a dictionary that converts each XSD node to its corresponding JSON Schema format. It is found that this method produces good translation results in most cases.

According to the study, there are several semantic principles must be followed to transform an XSD snippet into the corresponding JSON Schema. First, a list of data values is provided. Secondly, the form can only have a maximum of 5 values. Lastly, each value is an integer that is not negative. Furthermore, a genuine XML document requires a root element, and this element cannot be used to build one. A basic and acceptable JSON Schema can be created based on these assumptions. For the translation process, the procedures are followed as: For each XSD node, a translation dictionary is created to its matching JSON Schema. When traversing the XSD tree, the node's id and attribute are initially saved until there are no more child nodes. Then, using the dictionary, the node is translated from bottom to top. The data pieces now can be traversed into the appropriate JSON Schema format. Recursion can be used to explore some XSD elements with child nodes. In the process of navigating, there are three points to which need attention: first, each node is given a unique identity (ID), and its namespace and tag name are saved; second each XSD attribute's keyword names, values, related collections, and default inheritance attributes are recognized; third, identifying the element, store its parent node element, and identifying the textual information if the child element of an element is a simple text and there is no nested XML node. The element node can be transformed into its equivalent JSON data type if it has an XSD namespace and is a basic XSD data type. There are several specified data types in XML Schema. Although JSON has a restricted amount of data types, XSD can be used to confine these data types to these limitations. The XSD-to-JSON Schema type-conversion data dictionary is created by giving an equivalent JSON Schema for all declared XML data types. The complete translation process is traversed from the XSD node's root node to the smallest segment (child node), and the child node is then translated. As a result, the parent node gets translated, and the entire XSD document is finally converted to JSON Schema format.

Overall, by traversing the XSD document node and defining the data attribute dictionary for XSD to JSON Schema conversion, the technique may transform XSD format data into JSON Schema. Because JSON Schema provides fewer data characteristics, it's difficult to implement all the data dictionary's conversion rules, making it impossible to convert all XSD parts.

2.2 Existing Methods

The literature review on the subject yields two studies that are directly related to the scope of this report. They are briefly explained in this section.

2.2.1 From XML Schema to JSON Schema: Translation with CHR

Again, since it seems impossible to convert JSON and XML formats to each other without error, even though, Schema languages exist to specify the structure of sample documents for both data formats, Nogatz et. al. [5] introduce an application of a language translator. It uses Prolog and Constraint Handling Rules (CHR) to transform an XML Schema into a JSON Schema document. Simply, an XML Schema document is spread out into CHR constraints, which makes it possible to specify translation rules declaratively.

Nogatz identifies CHR as a declarative programming language that follows the constraint-based programming paradigm as an extension of the host language. One can define CHR rules to manipulate entries in a constraint store. That is, depending on the currently saved constraint, you can remove some or add a new constraint. Basically, the CHR extension adds three types of rules to the logical programming construct to control the contents of the constraint store. These CHR program rules are well known in the mathematical logic of computer programming. In addition to simplification and propagation rules, there is a mixed type called *simpagation*.

The translation process represented in the study has several steps. The first step is reading the XML Schema into Prolog, the output is a nested Prolog term. Then, the related constraints will be propagated by traversing the nested Prolog term recursively. Their unique identities and references to parent and child nodes keep their places. After that, the defaults should be added since the Prolog's XML parser will read explicitly set attributes only. With a CHR simpagation rule, some settings are also done, too. Then, the translation step comes. By following the constraints propagated referred in the study, translation from XSD fragments to JSON Schema is done. All have the same form. A single JSON constraint is conveyed that holds the XSD fragment's JSON Schema; the guard guarantees that all node and node attribute constraints are of the XML Schema namespace. There may be numerous JSON constraints with the same identifier since some node restrictions apply to multiple CHR rules. A simpagation rule merges them with the assistance of a self-defined merge JSON Prolog predicate. The last two steps include wrapping up the JSON Schema and cleaning. When the root element of the provided XSD has been translated and its JSON constraints have been combined, the previous phase ends. In addition, the globally declared type definitions are merged into the root's JSON constraint's definitions object. Finally, the constructed JSON Schema object is cleaned up.

Overall, the study proposes a language translator was developed to translate an XML Schema to a JSON Schema equivalent. Because the module was built from the bottom-up approach, it has a test framework and several test cases. It may not be feasible to implement all restricting semantics due to the unavailability of specific capabilities in JSON Schema. Another functionality that is lacking is the ability to handle linked XSD documents.

2.2.2 Towards Efficient and Unified XML/JSON Conversion

As we have already said, for data representation and/or exchange XML and JSON are the two most widely used formats When switching from one format to another for data representation, available converters can be used. Šandrih et. al. [7] describes a structure that converts XML and JSON formats to each other, which is compared to an existing and publicly available converter. Some differences and difficulties are found in the conversion when using different converters. Some results of the comparison are presented.

JavaScript is used to create the converter. Building a JavaScript object and deserializing it are the first two steps to converting from XML to JSON for data representation. The types of the child nodes of the input XML DOM tree of the document are inspected during the process of building JavaScript objects, which includes the types of elements, text, and CDATA sections. If a node is an element, its attributes are initially gathered as fields of the JavaScript object that corresponds to it. After that, the offspring of this element's simple fixture is checked. Objects are constructed in somewhat various ways depending on the quantity of text content and CDATA sections. If an element is not a leaf, the same function is run recursively for the subtree with this child element as the root. Instead of using a built-in function, the deserialization step is done with a custom function. Steps in the conversion of input JSON to the appropriate XML representation are converter independent. To parse a generic JSON string: First, the open tag () is written first, followed by the attributes-values pairs; if this element has children, the close tag (>) is inserted next, and the function is performed recursively using the children elements as arguments. The text and CDATA parts are left alone. The parent element's tag closes after a recursive call. If an element has no children, it is closed immediately (/>).

Also, two converters are compared in this study. In several cases, it is discovered that both converters suggest different conversions. These two options are available, based on personal preferences and logic. The following are some possible options: If CDATA sections are present in data and the content of these sections is relevant, one converter may be a better option. With one converter, data loss is conceivable. Similarly, with one converter, time climbs rapidly for huge files. Invertibility might be an essential criterion as well. A solution is offered in this study that would result in preserving most of the

information. It is managed to save the content, but in some cases, the order could not be restored, which is left as future work.

3 Results

4 Conclusions

The popularity and usefulness of both JSON Schema and XSD make them great candidates for data exchange and integrity checkers. However, converting one to another can be challenging due to the lack of standardization in JSON Schema. There are not many studies that provide such methods in the literature. The ones that provide also describe bidirectional conversion methods i.e. from XSD to JSON Schema or vice versa. The existing research has aged from 10 to 5 years which might be considered a little bit old. There is not much research on the utilization of trending machine learning topics to make the conversion. As future work, these subjects can be possible candidates. A strong example that proves those techniques work on this type of problem can be Github Copilot [2].

References

[1] David C Fallside and Priscilla Walmsley. "XML Schema Part 0: Primer Second Edition". In: *October* October (2004), pp. 1–84.

[2] *GitHub Copilot · Your AI pair programmer.*

[3] Shijiao Guo, Hongxia Xia, and Guangli Xiang. "Research on the translation from XSD to JSON schema". In: *2017 9th IEEE International Conference on Communication Software and Networks, ICCSN 2017*. Vol. 2017-Janua. Institute of Electrical and Electronics Engineers Inc., Dec. 2017, pp. 1393–1396. ISBN: 9781509038220. DOI: 10.1109/ICCSN.2017.8230338.

[4] *JSON Schema | The home of JSON Schema.*

[5] Falco Nogatz and Thom Frühwirth. "From XML Schema to JSON Schema: Translation with CHR". In: 2 (June 2014).

[6] Felipe Pezoa et al. "Foundations of JSON schema". In: *25th International World Wide Web Conference, WWW 2016*. International World Wide Web Conferences Steering Committee, 2016, pp. 263–273. ISBN: 9781450341431. DOI: 10.1145/2872427.2883029.

[7] Branislava Sandrih, Dusan Tosic, and Vladimir Filipovic. "Towards Efficient and Unified XML/JSON Conversion - A New Conversion". In: *Ipsi Bgd Transactions on Internet Research* 13.1, SI (2017), pp. 58–64. ISSN: 1820-4503.