2023 International Conference on Computational Science and Computational Intelligence (CSCI) | 979-8-3503-6151-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSCI62032.2023.00258

Converting WOX Objects to YAML Documents

Vivian P. Reynoso-Sánchez, Betzabet García-Mendoza, Carlos R. Jaimez-González, Wulfrano A. Luna-Ramírez
Departamento de Tecnologías de la Información
Universidad Autónoma Metropolitana, Unidad Cuajimalpa
Mexico City, Mexico
Email: {vivian.reynoso, bgmendoza, cjaimez, wluna}@cua.uam.mx

***Abstract*—Interoperability refers to the capability of enabling communication between applications developed in different programming languages. This communication is achieved by exchanging data, typically using a standardized format. To solve the challenge of interoperability, one approach involves the implementation of serializers that transform objects into standard formats, which can subsequently be transformed back into any programming language. This paper introduces a converter as a solution to tackle the interoperability issue, particularly focusing on text-based formats. The converter presented in this paper allows to transform objects written in the XML format generated by WOX to documents written in the YAML format. The paper explores object representations in WOX and YAML, and provides examples of the transformations obtained. A comparative analysis of some existing tools similar to the developed converter is carried out and their most relevant features are described.**

Keywords—converters, interoperability, programing languages, text-based formats, WOX objects

I. Introduction

Serialization refers to the procedure of transforming an object into a format suitable for storage in various mediums, including memory, files, databases, or streams, allowing for its transmission over a network. The main objective of serialization is to preserve an object's state so that it can be reconstructed at a later time when needed. The opposite operation, which restores an object from its serialized form, is known as deserialization.

Several programming languages offer built-in support for serialization, either as an integral part of the language itself or via an associated library. This library or software component is typically referred to as a serializer. In an ideal scenario, the serialization and deserialization capabilities inherent to programming languages would suffice for representing the state of an object that contains data types and fundamental language structures. However, in some programming languages, serialization is limited, as it cannot fully capture more intricate language constructs present in the object's state, such as object references, collections, enumerations, and similar complexities.

Serialization and deserialization are common procedures employed in distributed systems and applications that require data exchange. An example of this situation is transmitting an object to a remote application via a web service or through a remote procedure call. Additionally, certain applications demand the exchange of objects between different programming languages. In such cases, it becomes imperative to establish a language-neutral representation of objects, essentially a standardized format for describing serialized objects.

A programming language-independent representation of objects is a complex task to achieve, for several reasons: 1) objects need to be accurately reconstructed, considering both single and multiple inheritance; 2) complex data structures must be carefully restored, especially those where an object may be referenced multiple times by different pointers or references; 3) collections of objects, such as lists and dictionaries, must be restored in a suitable manner; 4) the size of numeric data types must be managed correctly, among other challenges.

When it comes to formats for representing serialized objects, they can be broadly categorized into two main groups: text-based formats and binary formats. Examples of widely used text-based formats for object representation include the Extensible Markup Language (XML) [1], JavaScript Object Notation (JSON) [2], and YAML Ain't Markup Language (YAML) [3]. Binary formats, on the other hand, are more reliant on their specific implementation and programming language, making them non-standardized. Text-based formats are designed for human comprehension, enabling manual inspection and often simplifying portability across different programming languages. However, it's worth noting that serializing objects into text generally requires more time and storage space. In terms of their practical use, all three mentioned text-based formats share a common purpose: providing a standardized means for representing structured data and a mechanism for data exchange that is independent of the programming language.

Interoperability essentially refers to the capacity for applications written in distinct programming languages to communicate with each other. This communication is achieved by exchanging data, typically in a standardized format. Resolving interoperability challenges can be accomplished by creating serializers that transform objects into standard formats, which can subsequently be deserialized in any programming language. It's worth mentioning that the third author of this paper has previously addressed this issue using this approach.

In this paper, a converter is introduced as a solution to address the interoperability challenge, with a specific emphasis on text-based formats. The converter presented in this paper enables the transformation of objects initially written in the XML format generated by the Web Objects in XML (WOX) serializer into documents formatted in YAML. The WOX serializer will be briefly described in Section II.

The rest of the paper is structured as follows. In Section II, the Web Objects in XML (WOX) framework [4] is introduced, which is the background of the converter developed; this section also shows the mechanism employed by WOX, outlines the

2769-5654/23/$31.00 ©2023 IEEE
DOI 10.1109/CSCI62032.2023.00258

serialization and deserialization procedures for objects, introduces the WOX extensions, and presents research studies comparing resource usage and performance in applications utilizing JSON, XML, and YAML for data exchange. Section III concentrates on describing YAML, detailing its data types and restrictions. Section IV provides a comparative analysis of some tools similar to the converter developed. Section V shows the converter and a series of objects in the XML format, and their corresponding YAML representation. Finally, Section VI presents some conclusions and future work.

II. Web Objects in XML

The foundation for the converter developed lies in the Web Objects in XML (WOX) framework [4], which was originally designed for the management of distributed objects and web services. WOX has evolved over the years, initially offering functionality solely for creating and managing remote objects and web services. Today, it encompasses significant features relevant to the field of distributed systems, particularly addressing issues of interoperability between different systems. It's worth noting that the WOX framework and its components have been systematically employed to support various courses within the undergraduate program in Information Technologies and Systems at our university. In particular, the WOX framework and its components have been used in the following courses: Dynamic Web Programming, Systems Integration, Object-Oriented Programming, and Thematic Laboratories.

WOX operates as a framework that employs the HTTP protocol to facilitate communication between clients and servers. It relies on XML as the chosen format for representing objects and ensures the availability of objects through unique identifiers (URLs). This design is influenced by the principles of the Representational State Transfer (REST) architectural style [5]. This section provides a concise overview of selected features and capabilities offered by the WOX framework.

The mechanism used by WOX for invoking a method on a remote object is depicted in Figure 1, and the steps involved are the following: 1) the client program initiates a method call on a remote reference, using the same method-calling approach as it would for a local object; 2) the WOX dynamic proxy intercepts the request, converts it into XML, and transmits it over the network to the WOX server where the remote object is situated; 3) upon receipt of the request, the WOX server deserializes it into a WOX object; 4) the WOX server loads the object into memory and executes the requested method; 5) the result of the method execution is sent back to the WOX server; 6) the WOX server serializes the result into XML, returning either the actual result or a reference to it to the client program (in case a reference to the object has been transmitted, the result is stored on the server); 7) the WOX dynamic proxy receives the result and deserializes it into the corresponding object, which could be a real object or a remote reference; 8) finally, the dynamic proxy delivers the result back to the client program.

The process of converting an object into XML (serialization) and converting XML into an object (deserialization) is executed by the WOX serializers [6, 7]. Initially, these serializers were developed for the Java and C# programming languages. Consequently, they enable the serialization of Java objects into XML, the deserialization of XML into C#, and vice versa. These WOX serializers work as standalone libraries, capable of generating XML representations of objects in a language-independant format. They are freely accessible for download [8]. It's worth mentioning that an article was published in Microsoft's MSDN magazine [9], discussing interoperability between Java and .NET applications. In this article, WOX serializers were employed to exchange objects between their respective applications; the article highly recommends the use of WOX for integrating heterogeneous applications.

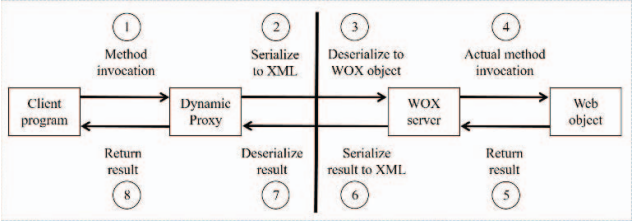

Fig. 1. WOX mechanism for invoking a method on a remote object.

In order to serialize objects into XML, WOX serializers employ the following process:

1) The process begins by extracting the name, type, and value of each object attribute. This is achieved using reflection, a programming capability that enables the inspection and potential modification of a program's high-level structure. Through reflection, it becomes possible to access an object's information, including the execution of its public attributes and methods, all during runtime. Additionally, introspection is utilized to determine the data type of an object attribute.

2) Once the name and value of each attribute within the object intended for serialization are acquired, they are then written into an XML document. If the value is not a primitive type but instead another object, all attributes of this subordinate object must also be represented within the XML document.

The code snippet displayed in Figure 2 illustrates the XML representation of an object belonging to the *Product* class after undergoing serialization with a WOX serializer. At the root of the XML document lies an *object* element, annotated with a *type* attribute set to *"Product"*, indicating the class to which the object belongs. The *id* attribute serves the purpose of managing references to objects, though in this specific instance, only one object is involved. Each attribute within the object is depicted through *field* elements, each equipped with the following attributes: *name* (reflecting the attribute *name* in the class), *type* (indicating the WOX data type of the attribute), and *value* (the attribute's value for that specific object). In WOX, primitive types are presented as *field* elements. The serialization rules of WOX serializers, along with several examples of serialized objects, are explained in [7] and [8].

```
<object type="Product" id="0">
  <field name="name" type="string" value="Corn"/>
  <field name="price" type="double" value="3.98"/>
  <field name="grams" type="int" value="500"/>
  <field name="reg" type="boolean" value="true"/>
  <field name="categ" type="char" value="\u0041"/>
<\object>
```

Fig. 2. XML representation of object belonging to a *Product* class.

The deserialization process is the inverse of serialization. It unfolds in the following sequence of steps: 1) information pertaining to the object contained within the XML document is extracted; 2) a class is created with the information extracted from the XML document; 3) in the relevant programming language, an object is instantiated, utilizing the information extracted from the XML document and the class generated in step 2. The WOX serializers [7] execute this process.

The WOX framework had an expansion through the addition of asynchronous communication capabilities [10], which are employed for processes that demand a significant amount of time to be completed on the server. With asynchronous communication, a client program can persist in its operations without becoming blocked, even while the server-side process is in progress. Once the process concludes its execution, the client can then access the process result. Within this context, WOX serializers continue to play a crucial role, as they handle the serialization (storage) and subsequent deserialization of these results. Furthermore, an asynchronous methods monitor [11] was conceived and developed to empower client programs to oversee the ongoing processes on the server. This monitoring tool can be accessed from a client program or via a web browser.

To further advance the research outlined thus far, a research project on interoperability in object-oriented programming languages was carried out, which led to the ability to inspect and navigate objects via a web browser [12]. This capability was made possible because WOX objects are stored on the server through their unique URLs. With these tools, it is posible to visualize the XML document representing a specific object or a portion thereof by navigating the object using XPath expressions [13]. Similarly, the web browser interface permits the display and execution of methods associated with an object stored on a WOX server [14]. Additionally, two serializers and deserializers were conceived and implemented in the Python [15] and PHP [16] programming languages. Furthermore, two websites were constructed to provide information and examples of their usage; these resources are also available for free download [17, 18].

The research work discussed so far provides the foundation for the XML to YAML converter outlined in this paper. Additionally, numerous studies emphasize the significance of data exchange between applications written in diverse programming languages, running on various devices and platforms. This underscores the importance of representing objects or data in a standardized format like JSON, XML, or YAML to ensure consistent communication across different applications and devices. Currently, these text-based formats, namely JSON, XML, and YAML, are among the most commonly employed formats for data exchange.

Studies conducted in [19, 20, 21] have compared the resource usage and performance of applications that employ JSON, XML, and YAML for data exchange. These studies have found that JSON and YAML formats consume less memory than XML when representing objects, and they also have faster object serialization times compared to XML. Additionally, [22] explores object serialization in both XML and JSON using different libraries. It involves serializing a sample object with these libraries and measuring the resulting file size and serialization time. The conclusion is that no single solution is definitively better than the others, as each library is suitable within its intended context. Moreover, both JSON and XML offer interoperability across different programming languages. [23] provides a review of the process of serializing objects into JSON format and vice versa. It emphasizes that the JSON format is more efficient than the XML format due to its smaller file size and quicker serialization process.

The research discussed in [24] focuses on analyzing XML and JSON within the context of decision-making computational systems. The study concludes that both technologies come with their own set of advantages and disadvantages. It suggests that for applications using simple data structures, JSON is a more suitable choice than XML. However, for applications with complex structures, XML is the preferred option. In comparative studies conducted in [25, 26], the results consistently show that JSON and YAML formats outperform XML in terms of size and serialization time. Nevertheless, it's worth noting that in certain applications requiring the transmission of intricate data structures, XML proves to be more effective for representing data that cannot be adequately conveyed using JSON or YAML. Furthermore, as part of the future research direction, the development of converters between XML and JSON, XML and YAML, JSON and YAML, and vice versa, is proposed.

The following sections concentrate in the WOX (XML) to YAML converter. In particular, the YAML language with its different data types, a comparative analysis of tools similar to the XML to YAML converter, and the representation of XML objects with their corresponding YAML documents.

III. YAML AND DATA TYPES

YAML is a data serialization language, which is human-readable and easy to understand. It can also be used in conjunction with other programming languages and is often used to write configuration files [3]. YAML is often correlated with XML, but YAML is not a markup language, it is an interoperable, flexible and object-oriented language. On the other hand, XML is an extensible markup language that was designed to be backward compatible with the Standard Generalized Markup Language (SGML), therefore it had many design restrictions that YAML does not have; for example, XML is designed to support structured documentation, YAML is more focused on data structures and messaging [1].

YAML supports various data types such as arrays, dictionaries, lists and scalars; the format of YAML is case sensitive and space sensitive. YAML files have the .yml extension. Additionally, the following notation --- is used to indicate the start of a YAML file. The data mapping syntax consists of a key followed by its value (key: value); it is very important to take the space into account since it is recognized. YAML can recognize some data types such as strings, characters, integers, floats, booleans, arrays, and lists that are constructed from primitive data types. Some examples of data mapping in YAML are shown in the following paragraphs.

In YAML it is possible to store primitive data types such as *int*, *float*, *string*, *boolean* and *NULL*. Figure 3 shows the representation of these types of data: the first attribute or key is *name*, which is a *string* and must be enclosed in single or double quotes; *age* value is an *int* data type; *height* is a *float* data type;

married is a *boolean* data type with two posible values (*true* or *false*); and *children* has the *NULL* value.

```
---
name: "Daniela"
age: 23
height: 1.68
married: false
children: NULL
```

Fig. 3. YAML representation of primitive data types.

Multiline and single-line strings are also possible in YAML. The ">" symbol allows to write a single-line string on multiple lines. The string is simply a long string, which is divided into several lines. With this notation the interior line breaks are eliminated. Figure 4 shows an example. It should be noted that another way to write in block is using the "|" symbol instead of the ">" symbol; which will also allow to have multiple line strings, but line breaks are preserved.

```
---
About:>
  Hello my name is Carlos
  I am from Mexico City and I
  like football
```

Fig. 4. YAML representation of single-line and multiline strings.

A list in YAML is a sequence of objects, it is an ordered collection of values. These values are not associated with a key, but with a positional index obtained from the order in which they are specified in the list. Figure 5 illustrates the way in which the lists are represented in YAML. A list can contain any number of elements, each element in the list is indented by a number of spaces and is preceded by the "-" symbol.

```
---
Frutas:
 - manzana
 - uva
 - mango
 - papaya
```

Fig. 5. YAML representation of a list.

YAML represents an array as a group of values arranged in sequence. Figure 6 shows an example of an array of four elements: *element1*, *element2*, *element3* and *element4*. The array can also be represented as shown in Figure 7.

```
---
array:
 - element1
 - element2
 - element3
 - element4
```

Fig. 6. YAML representation of an array.

```
---
array: ['elem1', 'elem2', 'elem3', 'elem4']
```

Fig. 7. Another YAML representation of an array.

Objects in YAML are represented as a group of key-value pairs. Figure 8 shows some valid expresions of *key: value* pairs.

```
---
Germany: Berlin
Spain: Madrid
France: Paris
Italy: Rome
```

Fig. 8. YAML representation of some key-value pairs.

Data types represent a problem when talking about interoperability between various programming languages. There must be an agreed upon mapping for each data type in a certain programming language. A mapping table is required, which contains the data types accepted by YAML and WOX. The first and second columns of Table 1 show the data mapping from WOX to YAML, and the third and fourth columns of Table 1 show the data mapping from YAML to WOX.

TABLE I. DATA MAPPING BETWEEN WOX AND YAML

WOX	YAML	YAML	WOX
byte	int	int	int
short	int	float	float
int	int	string	string
long	int	boolean	boolean
float	float	list	list
double	-	map	map
char	string	array	array
boolean	boolean		
string	string		
object	-		
array	array		
list	list		
map	map		
class	-		

It should be noted that there are some data types that exist in WOX, but does not exist in YAML. In particular, the *byte*, *short* and *long* data types do not exist in YAML, but the *int* data type could be used, which is an equivalent; the *char* data type does not exist in YAML, but the *string* data type could be used, which is an equivalent; the *array* data type in YAML is a sequence, it is represented as a *list*; the *map* data type in YAML is represented with *key-value* pairs.

IV. EXISTING TOOLS TO CONVERT XML TO YAML

This section describes the functionality and features of five different existing tools that are similar to the converter developed. At the end of the section there is a comparison table with relevant features of the tools analyzed.

Online YAML Tools [27] is a website that contains a XML to YAML converter. In order to use the converter the user enters

the XML in the input box positioned on the left and then the YAML will be obtained in the output box on the right. There is also the option to upload a XML document and download the YAML document that is generated. The website also contains examples of XML and YAML conversion, and it has an option to control the indentation desired for the YAML output.

JSON Formatter [28] is a website that helps to convert XML data to YAML. In order to convert the data the user has to write the XML in the left box and the YAML is automatically generated. It is also possible to upload a file to the website and obtain the corresponding YAML. In addition, an option is included to validate the XML entered, it is possible to download or print the YAML that is obtained. The wesbsite includes some examples of XML documents to convert them to YAML.

DocConverter [29] is a Java library that allows converting XML, JSON, CSV and YAML formats. This library contains the *DocConverter* class that can be called statically; this class has different methods. In order to use the XML to YAML format converter, it is needed to add the dependency to Maven and import the class. The *convertXmlToYaml(String xml)* method converts from XML to YAML, it receives a *string* as a parameter, which will contain the XML to be converted. In order to save the generated YAML the *convertStringToFile(String path, String content)* method is used, this method receives two parameters: the path where the YAML file will be saved and the *string* that contains the XML to be converted.

Browserling [30] is a website that hosts a simple XML to YAML converter, all the user needs to do is to type or paste the XML into the text field and press the button to get the YAML generated. The website has an input text box to write the XML text that will be converted to the YAML format; there are two buttons, one to convert from the XML format to the YAML format, and the other to copy the text to the clipboard.

Xmlplain [31] is a module in the Python programming language that converts XML to YAML and viceversa; it only allows to convert simple data types such as lists, dictionaries and strings. This module can be used by importing it in Python, in order to convert the data from XML to YAML. The XML file is read and there is a function that return YAML; it is also possible to write the generated YAML to a file.

Table 2 shows a comparison of the tools analyzed in this section: T1) Online YAML Tools, T2) JSON Formatter, T3) DocConverter, T4) Browserling, and T5) Xmlplain. A tick indicates that the tool has the feature, while a cross indicates that the tool does not have it. The features considered for the comparison are explained in the following paragraph.

The *web application* feature indicates that the tool can be accessed through a web browser; the *interoperability* feature refers to the ability of the tool to exchange information, whether data, documents or objects between different programming languages; the *YAML saving* feature refers to whether the tool has the ability to save the generated YAML; the *validate XML* feature indicates that the tool is able to check that a document in XML format is well formed and conforms to a specific structure; the *examples* feature means that the tool has examples about XML and YAML; the *upload file* feature refers to whether the user can select a document hosted on the computer to be uploaded to the tool; the *input text* feature refers to the existence of a box that allows the user to enter or paste text; the *structured YAML* feature means that the elements of a YAML document must follow a structure and have an indentation.

TABLE II. COMPARISON OF FEATURES OF THE ANALYZED TOOLS

Features	T1	T2	T3	T4	T5
Web application	✓	✓	×	✓	×
Interoperability	×	×	✓	×	✓
Save YAML	✓	✓	×	×	✓
Validate XML	×	✓	×	×	×
Examples	✓	✓	×	×	×
Upload file	✓	✓	✓	×	✓
Input text	✓	✓	×	✓	×
Structured YAML	✓	✓	×	✓	✓

V. OPERATION OF THE WOX TO YAML CONVERTER

After analyzing the different tools and their features shown in the comparison table, and analyzing the data types supported by WOX and YAML through the mapping table presented in a previous section, a web application was designed and developed, which contains a converter of objects from the WOX format (which is a specific XML format) to the YAML format. This converter makes it possible to achieve interoperability between applications that use WOX as the format to exchange data and applications that use YAML format.

The web application developed has all the features described in Table 2: it is a web application, the converter allows interoperability between WOX and YAML, it allows to save the YAML document in a file, it validates the XML input, it will contain several examples to convert different types of WOX objects to YAML documents, it is possible to upload XML files to the web application, it has an input text to write the WOX object, and it generates a structured YAML.

Figure 9 shows a screenshot of the web application developed, which has two input boxes: the left input box is to write the WOX object to be converted, while the right input box is to display the conversion to a YAML document. The interface shows three buttons: the *Load file* button is to allow the user to open a WOX file and load it to the left input box; the *WOX to YAML* button is to execute the conversion from the WOX format to the YAML format; and the *Download YAML* button is to dowload the resulting YAML as a document.

It should be noted that in Figure 9 the WOX object written in the left input box is an object of the *Course* class, which contains three attributes: *code* with a value of *6756*, *name* with a value of *"XML Tech"*, and *term* with a value of *3*; the data types of these attributes are *int*, *string* and *int*, respectively. The YAML document shown in the right input box is the result of the conversion, and it represents the WOX object in YAML.

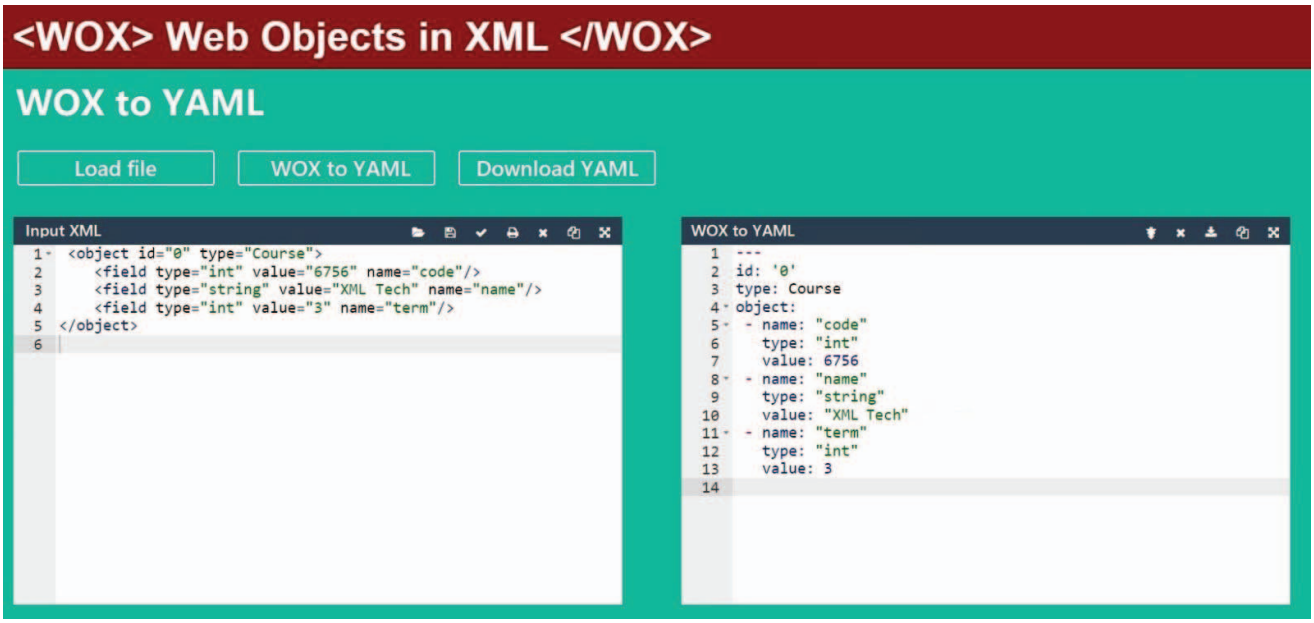

Fig. 9. The web application developed with the WOX to YAML converter.

Figure 10 shows another XML document that represents an array of two *Product* objects in WOX format. Every *Product* object has five attributes: *name*, *price*, *grams*, *registered* and *category*. The WOX data types of the attributes are as follows: *string*, *double*, *int*, *boolean* and *char*.

```
<object id="0" length="2" elementType="Product"
   type="array">
  <object id="1" type="Product">
       <field type="string" value="Baked beans"
           name="name"/>
       <field type="double" value="1.75"
          name="price"/>
       <field type="int " value="250"
          name="grams"/>
       <field type="boolean" value="true"
          name="registered"/>
       <field type="char" value="\u0042"
          name="category"/>
  </object>
  <object id="2" type="Product">
       <field type="string" value="Basmati Rice"
           name="name"/>
       <field type="double" value="3.89"
          name="price"/>
       <field type="int" value="750"
          name="grams"/>
       <field type="boolean" value="true"
          name="registered"/>
       <field type="char" value="\u0052"
          name="category"/>
  </object>
</object>
```

Fig. 10. An array of *Product* objects in WOX format.

Figure 11 shows the result of converting the WOX object to YAML using the web application developed. The YAML document generated contains the array of *Product* objects.

```
elementType: 'Product'
id: 0
length: 2
type: 'array'
object:
  - id: 1
    type: 'Product'
    object:
      - name: 'name'
        type: 'string'
        value: 'Baked beans'
      - name: 'price'
        type: 'double'
        value: 1.75
      - name: 'grams'
        type: 'int'
        value: 250
      - name: 'registered'
        type: 'boolean'
        value: 'true'
      - name: 'category'
        type: 'char'
        value: '\u0042'
  - id: 2
    type: 'Product'
    object:
      - name: 'name'
        type: 'string'
        value: 'Basmati Rice'
      - name: 'price'
        type: 'double'
        value: 3.89
      - name: 'grams'
        type: 'int'
        value: 750
      - name: 'registered'
        type: 'boolean'
        value: 'true'
      - name: 'category'
        type: 'char'
        value: '\u0052'
```

Fig. 11. YAML document after converting the WOX object.

VI. Conclusions

This paper introduced a WOX to YAML converter, which allows to translate objects written in the XML generated by WOX to YAML documents, allowing interoperability between applications that use WOX and applications that use YAML to exchange data. The converter presented in this paper resides in a web application that was implemented specifically for this purpose, its functionality allows to save the YAML documents generated in files, it validates the XML input, it is possible to upload XML files to the web application, it has an input text to write the WOX object, and it generates a structured YAML.

This paper also provided an analysis of five existing tools that translate XML to YAML, and described some of their relevant features. The converter presented in this paper translates XML to YAML, taking into consideration the existing data types in WOX and YAML in order to maintain the interoperability between these two text-based formats. The paper also gave an overview of the WOX framework, with its functionality and the tools that have been developed to extend it.

Further work is needed to complete a web site with documentation and examples to convert a variety of WOX objects to YAML documents. It is also planned to put the web application in a server in order to be available for users. There is also ongoing work to develop a WOX to JSON converter [32].

References

[1] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, "The Extensible Markup Language (XML) 1.0", 2013, available at: https://www.w3.org/TR/xml/

[2] Introducing JSON. http://www.json.org/

[3] YAML Ain't Markup Language (YAML) Version 1.2. (2021). https://yaml.org/spec/1.2/

[4] C. R. Jaimez-González, S. M. Lucas, "Implementing a State-Based Application Using Web Objects in XML", in: Meersman R., Tari Z. (eds) On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS. OTM 2007. Lecture Notes in Computer Science, vol. 4803, pp. 577-594, 2007, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-76848-7_40

[5] R. Fielding, "Architectural Styles and Design of Network-Based Software Architectures", PhD thesis, USA, 2000.

[6] C. R. Jaimez-González, S. M. Lucas, "Interoperability of Java and C# with Web Objects in XML", in Proceedings of the International Conference e-Society (ES 2011), pp. 518-522, Avila, Spain, March 2011.

[7] C. R. Jaimez-González, S. M. Lucas, E. López-Ornelas, "Easy XML Serialization of C# and Java Objects", Balisage: The Markup Conference 2011, Montréal, Canada, August 2011, in Proceedings of Balisage: The Markup Conference 2011, Balisage Series on Markup Technologies, vol. 7. https://doi.org/10.4242/BalisageVol7.Jaimez01

[8] C. R. Jaimez-González, S. M. Lucas, "Web Objects in XML (WOX): Efficient and easy XML serialization of Java and C# objects", http://woxserializer.sourceforge.net/

[9] I. Khan, "Interoperability: Runtime Data Sharing Through an Enterprise Distributed Cache", MSDN Magazine, vol. 25, No. 10, October 2010, http://msdn.microsoft.com/en-us/magazine/gg232763.aspx

[10] C. R. Jaimez-González, S. M. Lucas, "Asynchronous Method Invocations Using HTTP Polling and HTTP Streaming", in Proceedings of the International Conference on Applied Computing 2011 (AC 2011), pp. 536-540, Rio de Janeiro, Brazil, November 2011.

[11] C. R. Jaimez-González, W. A. Luna-Ramírez, S. M. Lucas, "A Web Tool for Monitoring HTTP Asynchronous Method Invocations", in Proceedings of the IEEE International Conference for Internet Technology and Secured Transactions, pp. 127-132, London, December 2012, https://ieeexplore.ieee.org/document/6470883

[12] C. R. Jaimez-González, "A Simple Web Interface for Inspecting, Navigating, and Invoking Methods on Java and C# Objects", Research in Computing Science: Advances in Computing Science, vol. 81, pp. 133-142, 2014, https://www.rcs.cic.ipn.mx/2014_81/RCS_81_2014.pdf

[13] J. Robie, D. Chamberlin, M. Dyck, J. Snelson, "XML Path Language (XPath)", W3C Recommendation, 2014.

[14] J. M. Hernández-Salinas, C. R., Jaimez-González, "Herramienta Web para Almacenar y Visualizar Objetos Distribuidos", Research in Computing Science, vol. 125, pp. 63-74, 2016.

[15] A. I. Rodríguez-Martínez, C. R. Jaimez-González, "Serializador de Objetos a XML en el Lenguaje de Programación Python", Avances de Ingeniería Electrónica 2013, pp. 444-451, 2013.

[16] L. Hernández-Piña, C. R. Jaimez-González, "Serialización de Objetos PHP a XML", Research in Computing Science, vol. 125, pp. 87-95, 2016.

[17] C. R. Jaimez-González, A. I, Rodríguez-Martínez, "Web Objects in XML in Python (PyWOX): Serializador de Objetos a XML en el lenguaje de programación Python", 2014, http://pywoxserializer.sourceforge.net/

[18] C. R. Jaimez-González, L. Hernández-Piña, "Web Objects in XML - PHP (PHPWOX): Serialización XML de objetos en PHP y viceversa", 2014, http://phpwoxserializer.sourceforge.net/

[19] N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study", in Proceedings of the 22nd International Conference on Computer Applications in Industry and Engineering (CAINE 2009), November 2009, San Francisco, California, USA.

[20] M. Ericksson, V. Hallberg, "Comparison between JSON and YAML for data serialization", BSc Thesis, Sweden, 2011.

[21] G. Goyal, K. Singh, K. Ramkumar, "A detailed analysis of data consistency concepts in data exchange formats (JSON & XML)", in Proceedings of the International Conference on Computing, Communication and Automation (ICCCA), May 2017, https://doi.org/10.1109/CCAA.2017.8229774

[22] K. Maeda, "Performance evaluation of object serialization libraries in XML, JSON and binary formats", in Proceedings of the Second International Conference on Digital Information and Communication Technology and its Applications (DICTAP), May 2012, Bangkok, Thailand, https://ieeexplore.ieee.org/document/6215346

[23] J. Mora-Castillo, "Serialización/deserialización de objetos y transmisión de datos con JSON: una revisión de la literatura", Tecnología en Marcha, vol. 29, No. 1, pp. 118-125, 2015, https://doi.org/10.18845/tm.v29i1.2544

[24] Z. Haq, G. Khan, T. Hussain, "A Comprehensive analysis of XML and JSON web technologies", New Developments in Circuits, Systems, Signal Processing, Communications and Computers. pp. 102-109, 2015.

[25] A. Breje, R. Gyorodi, C. Gyorodi, D. Zmaranda, G. Pecherle, "Comparative Study of Data Sending Methods for XML and JSON Models", International Journal of Advanced Computer Science and Applications (IJACSA), vol. 9, No. 12, pp. 198-204, 2018, https://doi.org/10.14569/IJACSA.2018.091229

[26] K. Grochowski, M. Breiter, R. Nowak, "Serialization in Object-Oriented Programming Languages", Introduction to Data Science and Machine Learning, IntechOpen, pp. 1-18, 2019, http://dx.doi.org/10.5772/intechopen.86917

[27] YAML to XML Converter. World's simplest YAML tool. Online YAML Tools. https://onlineyamltools.com/convert-yaml-to-xml

[28] Best YAML to XML Converter Online. Json Formatter. https://jsonformatter.org/yaml-to-xml

[29] GitHub - assimbly/docconverter: Java library to convert between XML, CSV, JSON and YAML documents. GitHub. https://github.com/assimbly/docconverter

[30] YAML to XML Converter - Transform YAML to XML. Browserling. https://www.browserling.com/tools/yaml-to-xml

[31] XMLplain. (2018). PyPI. https://pypi.org/project/xmlplain/

[32] C. R. Jaimez-González and B. García-Mendoza, "Towards the Development of Text-Based Format Converters for Object Representation," 2022 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2022, pp. 1879-1883, https://ieeexplore.ieee.org/document/10216513