2023 International Conference on Computational Science and Computational Intelligence (CSCI) | 979-8-3503-6151-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSCI62032.2023.00254

Web System for Visualizing and Executing Methods on Web Objects in XML

Betzabet García-Mendoza, José M. Hernández-Salinas, Carlos R. Jaimez-González
Departamento de Tecnologías de la Información
Universidad Autónoma Metropolitana, Unidad Cuajimalpa
Mexico City, Mexico
Email: {bgmendoza, cjaimez}@cua.uam.mx

***Abstract*—This paper presents a web system that allows to display and execute methods on objects created with the Web Objects in XML (WOX) framework, where the user can provide values for each of the method parameters through a web interface. The paper also reviews some existing systems with similar functionality and shows some of their relevant features. The interface of the system is presented, along with some examples that demonstrate its functionality, in particular, the paper shows how a user can display and execute methods on WOX objects. It should be noted that WOX is a framework that allows the creation of object-based distributed applications, which are interoperable among different object-oriented programming languages. WOX uses XML as the format representation for objects, and it uses HTTP as its transport protocol.**

Keywords—Web Objects in XML; visualization of methods, execution of methods, distributed objects

I. Introduction

The web system outlined in this paper serves as a complementary component to the Web Objects in XML (WOX) framework [1]. This framework is designed for the creation of distributed, object-oriented applications. WOX allows the development of distributed systems, employing XML as the format for representing objects and the messages exchanged between them [2]. Additionally, it offers both synchronous and asynchronous communication between clients and servers [3]. WOX incorporates unique features from two important paradigms in the development of distributed systems: the object-oriented approach and the web-based approach. Furthermore, WOX has the capability to store objects in Java [4], C#, Python [5], and PHP [6] programming languages, which can be created by either local or distributed applications.

The rest of the paper is organized as follows. Existing systems are presented in section 2, which have a similar purpose to the web system described in this paper. Section 3 provides an introduction to the WOX framework. Section 4 describes the actions that can be carried out by the developed web system, and shows some interface prototypes. Section 5 shows the web system in operation, in particular the visualization and execution of methods on objects are presented. Finally, section 6 discusses the conclusions and future work.

II. Existing systems

This section presents some systems that share similarities with the web system discussed in this paper. The systems under examination include the following: CORBAWeb [7], SopView+ [8], PESTO [9], CORBA Object Browser [10], and Apache Axis2 [11]. A more detailed comparative analysis of these systems is presented in [12], along with a concise overview of the features that were considered during the evaluation.

CORBAWeb is a system described in [7], which is an intermediary between the web and the Common Object Request Broker Architecture (CORBA). It is an object browser, designed to enable clients to inspect and execute methods on local or remote CORBA objects via a web browser. This system allows clients to navigate through CORBA object links using dynamically generated URLs for each remote object. CORBAWeb operates within a web browser interface, allowing users to access and execute methods on remote objects hosted on a server. In order to achieve this, it automatically generates HTML forms from the Interface Definition Language (IDL), facilitating the invocation of methods for any CORBA object. CORBAWeb interprets the user's actions, communicates with the required remote object to execute the object's method, retrieves the result, and ultimately delivers an HTML document containing the results of the method execution.

SOPView+ is a project detailed in [8], created within a UNIX environment and utilizing the Motif widget tool for constructing a graphical interface. The main objective of this project is to develop an object browser and viewer, primarily designed for querying and managing object-oriented databases. This system enables users to explore the database, locating their desired object, retrieve its information, and view it in a graphical format. Additionally, the tool organizes objects hierarchically and facilitates navigation within extensive databases by allowing users to select a base object, which serves as the starting point for navigation. SOPView+ allows users to modify the base object during their search for objects within databases by placing an anchor on the object. This feature simplifies the process of exploring objects in large databases.

PESTO is a system explained in [9] that originates from the GARLIC project [13], which aims to construct an information system capable of integrating data from various database systems. This data is accessible through a SQL-like language that incorporates object-oriented features. The GARLIC project's core objective is to introduce a fresh interface for querying and navigating objects, referred to as the Portable Explorer of Structured Objects (PESTO), which features its unique interface for object exploration and is tailored to interact

2769-5654/23/$31.00 ©2023 IEEE
DOI 10.1109/CSCI62032.2023.00254

with object databases, facilitating complex object queries within databases using a SQL-like language. This tool provides users with an interface that enables interactive navigation and content exploration of GARLIC databases. Similar to SOPView+, PESTO allows users to traverse the database both forward and backward. It allows users to query and navigate through objects presented in a hierarchical manner, with the ability to connect object nodes via links.

The CORBA Object Browser, described in [10], was designed to enable direct access to CORBA objects from a web browser using a URI scheme. It allows users to browse and invoke CORBA objects in a manner similar to how they navigate the Internet. Within this tool, users could view and execute the methods of a specific object directly from a web browser. However, it is important to note that this functionality required the use of a prototype browser known as the HotJava Web Browser, which is no longer available. The key advantage of the HotJava Web Browser was its capability to access secure CORBA objects hosted on a secure Object Request Broker (ORB) via the CORBA Object Browser. Accessing secure objects through the browser required authentication with the remote ORB and secure communication.

Apache Axis2, outlined in [11], is a web services engine developed by the Apache Software Foundation. This engine is designed for creating interoperable and distributed applications, implemented in both C++ and Java. Similar to WOX, Apache Axis2 is an open-source framework that relies on XML and SOAP for message exchange. Notably, Apache Axis2 operates with objects; however, it does not retain the state of these objects, resulting in its methods being invoked in a manner similar to static methods. An interesting characteristic of this tool is that it facilitates method execution on objects but lacks of a user interface. In order to access objects, users must call them via their respective URLs.

III. Web Objects in XML

This section offers a concise overview of the WOX framework, which merges characteristics of distributed object-oriented systems and distributed web-based systems. Some of the key attributes of this framework are introduced.

WOX employs URLs to provide unique identification for remote objects, aligning with the principles of the Representation State Transfer (REST) architecture [14]. This is a significant aspect because it means that all objects can be uniquely identified by their URL, allowing access from any location on the web, whether through a web browser or via programming.

WOX relies on an effective serialization tool known as the WOX serializer [2]. This serializer is the foundation of the framework, handling the serialization of objects, requests, and responses in client-server interactions. The WOX serializer is a standalone library that uses XML and has the capability to serialize objects from Java, C#, PHP, and Python into XML and vice versa. A notable feature of this serializer is its ability to generate standardized XML representations for objects, which are language-independent. This characteristic facilitates interoperability among various object-oriented programming languages, allowing applications written in these languages to work together seamlessly.

WOX features a collection of standard and unique operations used for interacting with both local and remote objects. These operations encompass actions such as requesting remote references, making calls to static methods (web service calls), invoking instance methods, object destruction, requesting copies, object duplication, updating and uploading objects, and invoking asynchronous methods, among other functions. Further details about some of these operations can be found in [1]. The mechanism used by WOX in a method invocation is explained as a series of steps:

Step 1. The WOX client program initiates the method invocation on a remote reference, much like invoking a method on a local object.

Step 2. The WOX dynamic proxy takes the request, converts it into XML format, and transmits it over the network to the WOX server.

Step 3. The WOX server receives the request and converts it back into a WOX object from XML.

Step 4. The WOX server loads the object and executes the method on it.

Step 5. The outcome of the method execution is returned to the WOX server.

Step 6. The WOX server transforms the result into XML, which is then sent back to the client, either as the actual result or as a reference.

Step 7. The WOX dynamic proxy at the client end receives the result and converts it into the appropriate object (real object or remote reference).

Step 8. The WOX dynamic proxy returns the result to the WOX client program.

From the perspective of the WOX client program, it merely initiates the method invocation and receives the result without being aware of the underlying process. The WOX client libraries handle the steps involving serialization of the request, transmission to the WOX server, reception of the method invocation result, and deserialization. The following sections will introduce the web system that facilitates the visualization and execution of methods on WOX objects via a web browser.

IV. Analysis and Design

This section describes the actions that can be carried out by the developed web system, and shows some of the interface prototypes to visualize and execute methods.

A. Actions

The actions that can be carried out by the web system are described in the following paragraphs.

A1) Access to a specific WOX object over the network. The web system provides users with a unique URL for each object, which the system generates. Using this URL, users can directly

reach the desired object without the need to navigate through the repository and search among all the objects stored in the same repository.

A2) Visualization of objects graphically. Users are allowed to view the attributes of a specific WOX object through an interface. There are two available viewing options: a) tabular view that presents the object in a table format, displaying all of its attributes, irrespective of their data type; b) a view that presents the object's XML code, which is formatted in a way that enables users to comprehend each XML tag.

A3) Storage of WOX objects on the server. This functionality allows users to upload WOX objects to the server, facilitating their later use as parameters for invoking objects or for straightforward storage in the repository.

A4) Visualization of methods of any WOX object. Users have the ability to inspect the methods associated with any WOX object, without concern for the parameters necessary for invoking each method. Each method is accompanied by a designated area where the response it generates upon invocation is displayed.

A5) Execution of any method belonging to the class of a WOX object. Users have the capability to execute methods on WOX objects, with a parameter validation mechanism in place to prevent the entry of incorrect parameter values. This ensures that method invocations are accurate. Additionally, the system can provide a response from the method invocation, regardless of its data type. It's worth noting that for the web system to execute a method on an object, it must possess the class to which that object belongs.

It should be noted that actions A1, A2 and A3, which have to do with access, visualization and storage of WOX objects, are covered in more detail in [12]; while actions A4 and A5, which have to do with visualization and execution of methods on WOX objects, are discussed in the following sections.

B. Interface prototypes

Figure 1 illustrates the interface prototype for the list of objects stored in the server's repository, which includes the following fields: *object URL*, which is a unique URL for direct access to a specific object within the repository; *name*, which displays the name of the XML file representing the WOX object; *reference*, which indicates the class to which each object is associated; *actions*, which offers two choices to users, the first option allows users to visually explore the object's attributes, while the second option allows to visualize and access the methods of an object.

Figure 2 shows the interface prototype for a method invocation; it shows the *object id*, the *class name* of the object, the *name* of the method to be invoked on that object, the *parameters* of that specific method with input boxes to enter the values, a button to execute the method, and an area to show the results of the method invocation.

Fig. 1. Interface prototype for the list of objects stored in the repository.

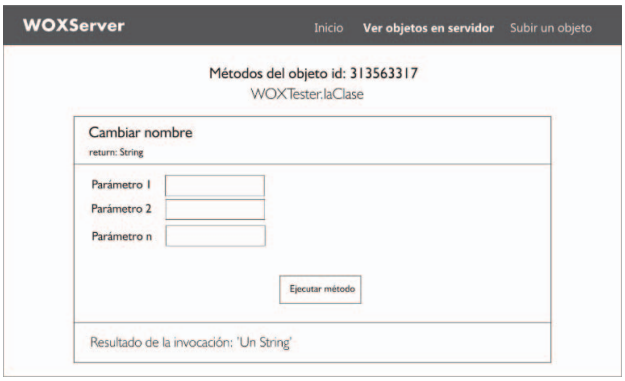

Fig. 2. Interface prototype for a method invocation.

V. Web System in Operation

This section presents the web system in operation. It describes how objects are visualized through the repository, how to visualize methods of an object, and how to execute methods on a particular WOX object.

A. Visualization of a WOX object

Figure 3 shows the operation of the web system to visualize a WOX object, through a series of steps that are carried out between client and server.

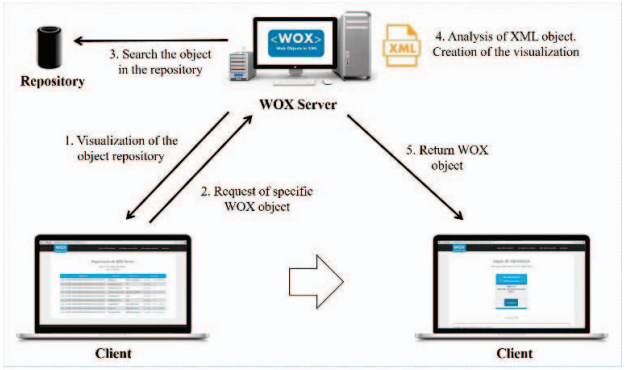

Fig. 3. Visualization of a WOX object.

Step 1. The user access the repository of objects, where every object is displayed with its URL, name, reference and actions (*view object* and *view methods*).

Step 2. The user selects the *view object* link from the *actions* for a specific object. The user can also acess the object directly with the URL that represents it.

Step 3. The system receives the *id* of the object, and proceeds to search for it in the repository. In order to do this, the *WOXRP.xml* object is deserialized and the location of the file containing the requested object (if it existed) is accessed.

Step 4. Once the location and name of the file have been obtained, the file is analyzed with the *viewObjectBYXML* method, which is responsible for creating an HTML table that will represent the graphical display of the WOX object, adding the *id* of the object, the *class* to which it belongs and the *attributes* it contains. Finally, it returns all the HTML code generated and displays it to the user.

B. *Visualization of the methods of a WOX object*

Figure 4 shows the operation of the system for visualizing all the methods of a WOX object, through a series of steps that are carried out between client and server.

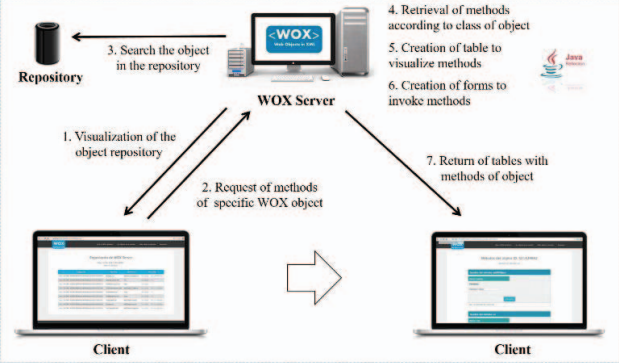

Fig. 4. Visualization of the methods of a WOX object.

Step 1. The user access the repository of objects, where every object is displayed with its URL, name, reference and actions (*view object* and *view methods*).

Step 2. The user selects the *view methods* link from the *actions* for a specific object.

Step 3. The system receives a request, which contains the *id* of an object to be manipulated, it proceeds to search for it in the object repository.

Step 4. If the object is found, the system obtains the class of the object and from the class, the system recovers all the methods and parameters necessary to invoke each method. In order to do this, the object uses Java Reflection technology.

Step 5. A table is created for each method (once all the methods and parameters of a specific class have been obtained), which will contain the following attributes: method name, type of response that it returns when the method is invoked (*object*, *int*, *float*, etc.), button to invoke the method, text field in which the response will be displayed.

Step 6. The system adds a form (once the table that represents each method has been generated), in which the user can provide the parameters required to invoke the method.

Step 7. Once all the method tables belonging to the class of an object have been created with their respective forms to be invoked, they are returned to the user.

C. *Invocation of a method on a WOX object*

Figure 5 shows the operation of the system for invoking a method on a particular WOX object, through a series of steps that are carried out between client and server.

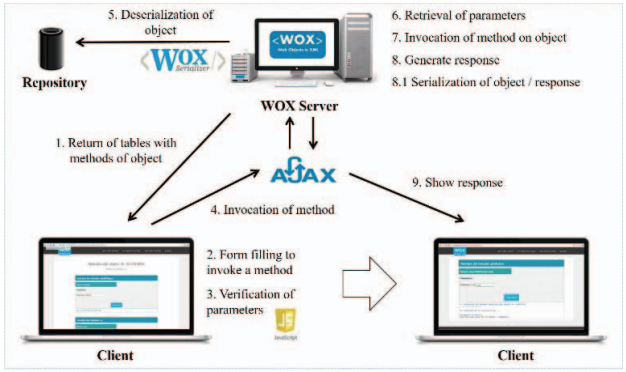

Fig. 5. Invocation of a method on a WOX object.

Step 1. The user receives the tables that represent the invocation of methods on a particular object.

Step 2. The user proceeds to fill out the form (parameter entry) to be able to invoke a certain method.

Step 3. During the filling out of the form, with the help of JavaScript, it is verified that the parameters entered by the user are correct. If an integer type number is required in the form for the method, it will be verified that the user really enters an integer, and not a decimal or text.

Step 4. After the user has entered the correct parameters to invoke a certain method, they can proceed to press the *test method* button, and with the help of an asynchronous JavaScript call (AJAX), the response is displayed just below the table that represents the method to be executed. Without having to access another tab or refresh the browser page, AJAX sends a request to the server and sends all the parameters entered by the user for the invocation of a certain method.

Step 5. The system receives the AJAX request with the following information: *id* of the object that contains the method to be invoked, *name* of the method to be invoked, list of parameters required to invoke the method, list of classes to which each parameter belongs. An example of the information that the server receives to invoke a method is the following:

- id: 3294242
- method name: “addBook”
- parameter list: 1, “book name”, “author”, “publisher”, true
- list of classes: int, String, String, String, boolean

Step 6. Once the information is obtained, the system locates the object according to the id provided, it is deserialized and the list of parameters is verified (the system verifies that each parameter matches the class).

Step 7. The system executes the requested method (once the list of parameters has been created and verified). In order to proceed with the method invocation, a new object called *answer* is created, which will store the result of the method execution.

Step 8. The system proceeds to verify the *answer* object (once the requested method has been invoked), for which it obtains its class. If the *answer* is a primitive object (*int*, *float char*, *string*, etc.) the result can be displayed on the screen as text. On the other hand, if the *answer* is a non-primitive object, it is serialized with WOXSerializer.

Step 9. The user is shown a link to view the object that will be the response to the invocation of the requested method.

D. *Web system interface*

In order to visualize the methods of an object, it is necessary that the object class is hosted on a WOX server. If this is the case, by pressing the *View methods* button the server will show the user all the methods of the object, with their corresponding fields to be invoked, as it is shown in Figure 6, which illustrates the *addBook* method of the *library* class.

Fig. 6. Interface to visualize the methods of a *library* object.

Figure 7 shows the *addBook* method, which will add a book to a *library* object, with the following values for its parameters:

- Book ID: 121211
- Name: LOST OCEAN
- Author: JOHANA BASFORD
- Editorial: CULTURAL DEVELOPMENTS
- Status: Available

The bottom panel shown in Figure 7 illustrates the results of the method invocation, the book has been added to the *library* object with the values specified. The results panel shows that the data type for the first parameter is *int*, and the data types for the rest of the parameters are *string*. There is also a message indicating that the book has been added correctly.

Fig. 7. Interface to invoke the *addBook* method on a *library* object.

Figure 8 shows the *addBook* method, which will add another book to the *library* object, with the following values for its parameters:

- Book ID: 121234
- Name: MEXICO DECEIVED
- Author: FRANCISCO MARTÍN MORENO
- Publisher: Planeta
- Status: Available

The bottom panel shown in Figure 8 illustrates the results of the method invocation, the book has also been added to the *library* object with the values specified.

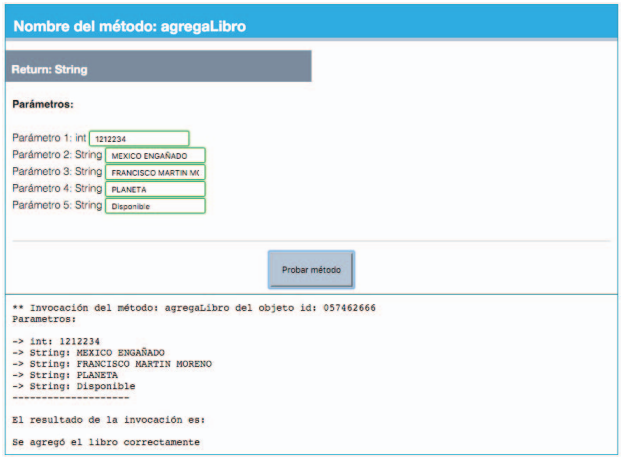

Fig. 8. Interface to invoke again the *addBook* method on a *library* object.

After adding two books to the *library* object through its methods, it can be displayed, as shown in Figure 9. It can be observed that the object has changed, it now contains the two books that were added previously.

Fig. 9. Interface to visualize the *library* object with the two books added.

VI. Conclusions and Future Work

This paper presented a web system that allows users to visualize and execute methods on distributed objects. The invocation of methods is carried out through a web interface, where the values for each of the parameters are provided. The web system developed is a complement of Web Objects in XML (WOX), which is a framework for programming distributed object-based applications.

The web system met the objectives set at the beginning of its development, since it was possible to successfully design and implement the visualization and invocation of methods on objects through a web browser. It should be noted that it is also possible to store objects in a repository, where they can be visualized, both graphically and in its XML representation.

References

[1] 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

[2] 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

[3] 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

[4] 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/

[5] C. R. Jaimez-González, A. Rodríguez, "Web Objects in XML (PyWOX): Object to XML Serializer in the Python programming language", https://pywoxserializer.sourceforge.net/

[6] C. R. Jaimez-González, L. Hernández, "Web Objects in XML (PHPWOX): Object to XML Serializer in the PHP programming language", http://phpwoxserializer.sourceforge.net/

[7] P. Merle, C. Gransart, J. Geib, "CorbaWeb: A Generic Object Navigator", http://www.lifl.fr/~merle/papers/96_WWW5/paper/Overview.html

[8] S. Chang, H. Kim, "SOPView+: An Object Browser Which Supports Navigating Database by Changing Base Object", in Proceedings of the 21st International Conference on Computer Software and Applications Conference (COMPSAC 97), 1997.

[9] M. Carey, L. Haas, V. Maganty, J. Williams, "PESTO: An Integrated Query/Browser for Object Databases", in Proceedings of the 22th International Conference on Very Large Data Bases, Mumbai, India, 1996, https://dl.acm.org/doi/10.5555/645922.673633

[10] G. Kumar, P. Jalote, "A Browser Front End for CORBA Objects", in 10th International World Wide Web Conference, 2001.

[11] Apache Software Foundation. Web Services - Apache Axis. http://ws.apache.org/axis/

[12] J. M. Hernández-Salinas, C. R. Jaimez-González, B. García-Mendoza, "Web System for Storing and Visualizing Web Objects in XML", 2022 International Conference on Computational Science and Computational Intelligence (CSCI 2022), Las Vegas, NV, USA, 2022, pp. 1914-1919, doi: 10.1109/CSCI58124.2022.00344

[13] M. Tork, M. Arya, L. Haas, M. Carey, W. Cody, R. Fagin, P. Schwarz, J. Thomas, E. Wimmers, "The Garlic Project", in Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, New York, 1996.

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