Online-Journals.org

iJET International Journal of Emerging Technologies in Learning

iJET | eISSN: 1863-0383 | Vol. 19 No. 4 (2024) | OPEN ACCESS

https://doi.org/10.3991/ijet.v19i04.46449

PAPER

Web System to Support the Teaching of an Undergraduate Distributed Systems Course

Carlos R. Jaimez-González(✉), José M. Hernández-Salinas, Betzabet García-Mendoza

Universidad Autónoma Metropolitana, Ciudad de México, México

cjaimez@cua.uam.mx

ABSTRACT

This paper introduces a web-based system that supports the teaching of an undergraduate (UG) distributed systems course. It specifically describes a web system that was developed to complement the functionality of web objects in the XML (WOX) framework. It allows for storing and visualizing the state of distributed objects, as well as displaying and executing methods through a web interface. Users can provide values for each of the parameters. The WOX framework is essential to note as it facilitates the development of distributed applications that are object-based and can interoperate among different object-oriented programming languages. WOX employs the XML format to represent objects and uses HTTP as the communication protocol.

KEYWORDS

distributed systems, undergraduate (UG) course, web objects in XML, teaching system, educational technology

1 INTRODUCTION

The web system introduced in this paper aims to support the teaching of an undergraduate (UG) distributed systems course. It specifically focuses on topics related to distributed objects and the interoperability of systems. The web system complements the web objects in the XML (WOX) framework [1], which was designed for building distributed, object-oriented applications. WOX enables the creation of distributed systems that use XML to represent objects and the messages exchanged between them [2]. It also supports both synchronous and asynchronous communication between clients and servers [3]. WOX combines key features from two important paradigms in distributed systems development: object-oriented and web-based approaches. Additionally, WOX offers the capability to interoperate with objects in various programming languages, including Java [4], C#, Python [5], and PHP [6], which can be generated by either local or distributed applications.

The rest of the paper is organized as follows: Existing systems with a similar purpose to the web system described in this paper are presented in Section 2. Section 3

Jaimez-González, C.R., Hernández-Salinas, J.M., García-Mendoza, B. (2024). Web System to Support the Teaching of an Undergraduate Distributed Systems Course. *International Journal of Emerging Technologies in Learning (iJET)*, 19(4), pp. 71–85. https://doi.org/10.3991/ijet.v19i04.46449

Article submitted 2023-11-03. Revision uploaded 2024-01-26. Final acceptance 2024-01-26.

provides an introduction to the WOX framework. Section 4 describes the functionality of the developed web system, presents the design of the object repository, and introduces the relevant classes. Section 5 presents the web system in operation. It illustrates the storage of objects, access to the repository, visualization of a WOX object, visualization of the methods of a WOX object, and the execution of specific methods. Finally, Section 6 discusses the conclusions and future work.

2 BACKGROUND

This section explores some systems that share similarities with the web system introduced in this paper. The analyzed systems include common object request broker architecture (CORBA) web [7], SOP View+ [8], portable explorer of structured objects (PESTO) [9], CORBA object browser [10], and Apache Axis2 [11]. A comparison of these systems is provided, along with a brief overview of the features that were considered during the analysis.

CORBA Web [7] is a system that serves as a bridge between the web and CORBA. It is an object browser designed to enable clients to inspect and execute methods on local or remote CORBA objects through a web browser. This system automatically generates HTML forms based on the interface definition language (IDL), simplifying the invocation of methods for any CORBA object. CORBA Web interprets user interactions, communicates with the necessary remote object to execute its method, retrieves the result, and presents an HTML document containing the method execution results.

SOP View+ [8] is a project that aims to create an object browser and viewer with a focus on querying and managing object-oriented databases. This system enables users to navigate the database of objects, locate their desired items, retrieve their information, and view them in a graphical presentation. Additionally, this tool organizes objects in a hierarchical structure and facilitates the exploration of objects within databases by enabling users to select a base object as the starting point for navigation. SOP View+ also allows users to modify the base object during their search for objects within databases by setting an anchor on the object.

PESTO [9] is a system that originated from the GARLIC project [12]. The GARLIC project was designed to develop an information system capable of integrating data from different database systems and making it accessible through a language similar to SQL but with object-oriented capabilities. This project provides an interface for querying and exploring objects known as the PESTO, which enables the exploration of objects in databases using a SQL-like language. PESTO is similar to SOP View+ as it supports querying and navigating through objects presented in a hierarchical structure.

The CORBA object browser [10] was developed to offer direct access to CORBA objects through a web browser using a URI scheme. It allowed users to browse and interact with CORBA objects in a way similar to how they navigate the Internet. In this tool, users can view and execute the methods of a specific object directly within a web browser. However, it is worth mentioning that this functionality relied on the use of a prototype browser called the Hot Java Web Browser, which is no longer available. The primary advantage of the Hot Java Web Browser was its capability to access secure CORBA objects hosted on a secure object request broker (ORB).

Apache Axis2 [11] is a web services engine designed to create distributed and interoperable applications. It supports implementations in both C++ and Java. It is similar to WOX because it is an open-source framework that relies on XML and SOAP for exchanging messages. Apache Axis2 works with objects, but it doesn't maintain the state of these objects, resulting in its methods being invoked in a way similar to static methods. An interesting feature of this tool is that it allows method execution

on objects but does not offer a user interface. In order to access objects, users need to call them using their corresponding URLs.

Table 1 provides a comparison of the systems examined in this section: S1) CORBA Web, S2) SOP View+, S3) PESTO, S4) CORBA Object Browser, and S5) Apache Axis2. In the table, a checkmark indicates that the feature is present in the system, while a cross indicates that the feature is not available. The features used for comparison are as follows: *open source* denotes whether the software is freely available, redistributable, and modifiable; *interoperability* indicates the system's ability to communicate across different platforms or programming languages; *based on objects* refers to whether the system supports remote objects; *web services* means that the system supports web services; *use of XML* denotes whether XML is used for communication between the client and the server; *visualization of objects* indicates whether the system allows graphical visualization of objects; *visualization of attributes* indicates whether the system allows the visualization of attributes within objects; *execution of methods* refers to the system's capability to execute methods of objects via a web browser; *web interface* denotes whether the system provides a web interface for users to view remote object methods; *database management* refers to the system's ability to navigate object databases or repositories.

Table 1. Features of the analyzed systems

Feature	S1	S2	S3	S4	S5
Open source	✓	×	×	✓	✓
Interoperability	✓	×	×	✓	✓
Based on objects	✓	✓	✓	✓	×
Web services	✓	×	×	×	✓
Use of XML	×	×	×	×	✓
Visualization of objects	✓	✓	✓	✓	×
Visualization of attributes	×	✓	✓	✓	×
Execution of methods	✓	×	×	✓	✓
Web interface	✓	×	×	✓	×
Database management	×	✓	✓	×	×

It should be noted that there have been other initiatives to support the teaching of distributed systems. These include a framework designed to assist students in distributed systems and computer networks courses building simulations and generating applets from algorithms or protocols [13], a teaching tool that utilizes virtualization for a parallel and distributed computing course [14], and a series of modules covering basic and advanced high-performance computing, as well as various parallel and distributed systems programming topics [15], among others.

3 WOX FRAMEWORK

This section provides an overview of the WOX framework, which combines features from distributed object-based systems and distributed web-based systems. Some of the features of this framework are presented.

WOX employs URLs to provide unique identification for remote objects, adhering to the principles of the representational state transfer (REST) architecture [16]. This is an important aspect because every object is uniquely identified by its URL and can be accessed from any location on the Internet, through a web browser or a program.

WOX uses an efficient serializer known as the WOX serializer [2]. This serializer forms the foundation of the framework for converting objects, requests, and responses shared between clients and servers. The WOX serializer is an independent XML-based library with the capability to serialize objects from Java, C#, PHP, and Python into XML and vice versa. One of the main advantages of this system is its ability to produce standardized XML representations for objects that are independent of any particular programming language. This feature facilitates interoperability among different object-oriented programming languages and applications developed using those languages.

WOX contains a variety of standard and specialized operations that are utilized with both local and remote objects. These operations involve tasks such as obtaining remote references, making calls to static methods (web service calls), invoking instance methods, removing objects, obtaining copies, replicating objects, updating and transferring objects, and calling asynchronous methods, among others. Detailed explanations of some of these operations can be found in reference [1]. The method invocation mechanism employed by WOX is illustrated in Figure 1.

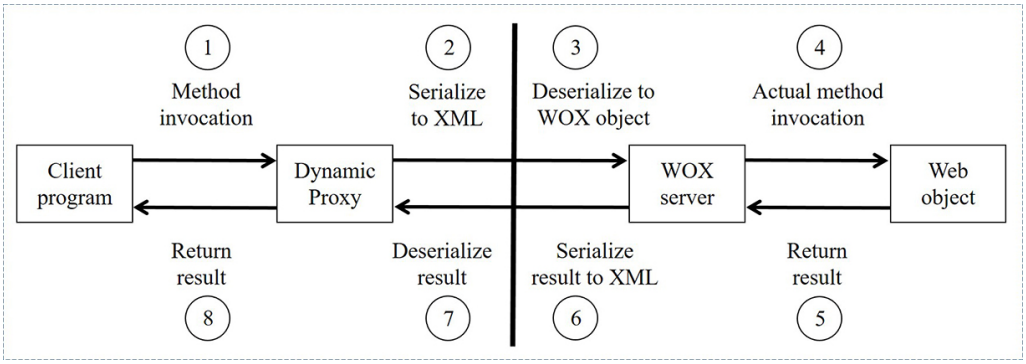

Fig. 1. Remote method invocation mechanism employed by WOX

The series of steps carried out in the invocation of a method in WOX is as follows: 1) the WOX client program invokes a method on a remote reference (the way in which the client invokes a method on a remote reference is exactly the same way as if it invokes a method on a local object); 2) the WOX dynamic proxy takes the request, serializes it to XML and sends it over the network to the WOX server; 3) the WOX server takes the request and deserializes it to a WOX object; 4) the WOX server loads the object and executes the method on it; 5) the result of the method invocation is returned to the WOX server; 6) the WOX server serializes the result to XML, and it is returned to the client, either the actual result or a reference to it (the result is stored in the server in case a reference is sent); 7) the WOX dynamic proxy receives the result and deserializes it to the appropriate object (real object or remote reference); 8) the WOX dynamic proxy returns the result to the WOX client program. From the perspective of the WOX client program, it only executes the invocation and receives the result back transparently. The WOX client libraries handle the serialization of the request, sending it to the WOX server, and receiving the result of the method invocation and deserialization. The following sections introduce the analysis, design, and operation of the developed web system.

4 ANALYSIS AND DESIGN OF THE WEB SYSTEM

This section describes the functionality of the developed web system, presents the design of the object repository, and introduces the relevant classes.

4.1 Functionality

The actions that can be performed by the web system are described as follows:

Access to a specific WOX object over the network. The web system provides users with a unique URL for each object, which is generated by the system. By using this URL, users can directly access the desired object without having to navigate through the repository and search among all the objects stored in it.

Visualization of objects graphically. Users can view the attributes of a specific WOX object through an interface. There are two available viewing options:

Storage of WOX objects on the server. This functionality enables users to upload WOX objects to the server, making it easier to use them later as parameters for invoking objects or for simple storage in the repository.

Visualization of methods for any WOX object. Users have the ability to inspect the methods associated with any WOX object without worrying about the parameters required to invoke each method. Each method is accompanied by a designated area where the response it generates upon invocation is displayed.

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 input of incorrect parameter values. This ensures that method invocations are accurate. Additionally, the system can provide a response to the method invocation, regardless of its data type.

4.2 Design of the repository

The developed web system includes an object repository where all WOX objects are stored. These objects can be either uploaded by users for manipulation or generated by the server itself through responses during method execution (e.g., when a method returns an object after execution or when a user retrieves an object contained within another object). Figure 2 shows the structure of the object repository.

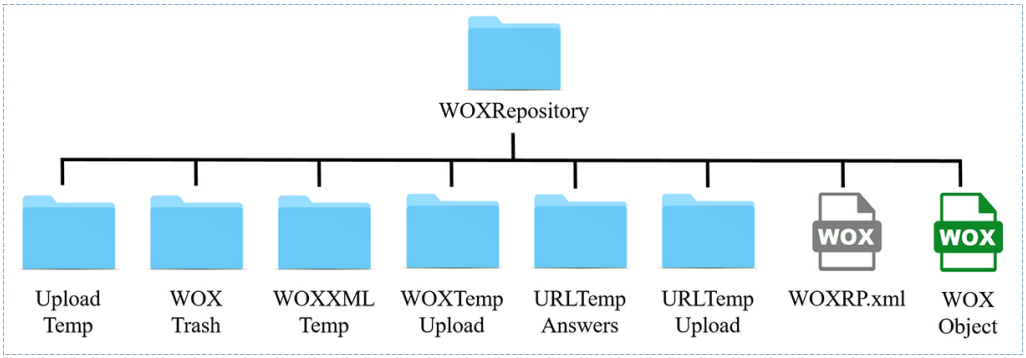

Fig. 2. Structure of the object repository

The WOX Repository is the primary repository where all the WOX objects uploaded by the user are stored. The other folders store new objects generated by the server.

There is a file called ***WOXRP.xml***, which is a serialized object that contains a list keeping track of every existing object in the main repository. This registry consists of the ID that uniquely identifies each object in the repository, the name of the XML file representing the object, and the class to which it belongs. ***WOX objects*** refer to all objects that exist in the main repository.

The Upload Temp folder is utilized to store objects uploaded by the user. The server must verify that the uploaded objects are indeed WOX objects and not

something else. Once the object is verified by the server, a record of the object is added to *WOXRP. XML*. If it was a WOX object, it is later transferred to the main repository; otherwise, it is moved to the trash folder (***WOX Trash***).

WOX Trash is a folder that represents the WOX server trash. It stores all the objects deleted by the user, old objects generated by the server during the execution of methods, and any incorrect objects the user attempted to upload.

When a user views an object that is contained within another object, the web system will extract that object and create a new one, which is stored in the folder ***WOX XML Temp*** for later use. When a user executes a method that returns an object, the system serializes it and stores it in the folder ***URL Temp Answers*** for later display to the user. When a user executes a method and includes a file containing a WOX object as a parameter, the web system saves the received file in the folder for later processing and to return a result. When a user executes a method and includes a URL containing a WOX object as a parameter, the web system will extract the code, create the object, and store it in the folder ***URL Temp Upload*** for further processing.

4.3 Class diagram

This subsection presents the class diagram for the web system, as shown in Figure 3. The classes involved in displaying the attributes and methods of an object are *WOX Server*, *WOX Visualizer,* and *WOX Tab Generator*. The classes responsible for invoking methods on objects are *WOX Server* and *WOX Invoker*. Lastly, the classes involved in managing the repository of objects are *WOX Object* and *WOX Server*. A description of these classes is provided in the following paragraphs.

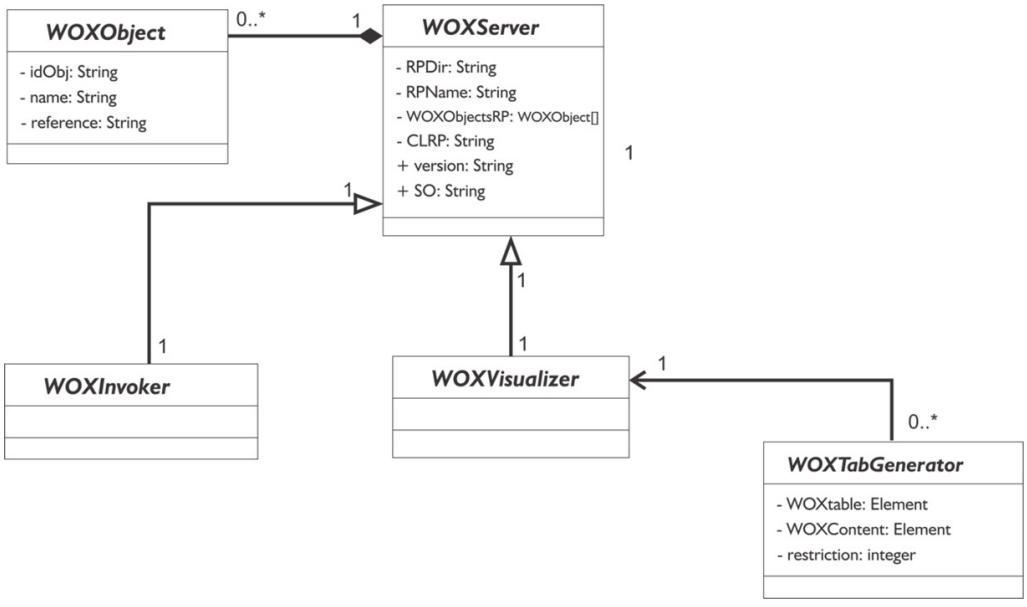

Fig. 3. Class diagram for the web system

The WOX Server is the main class of the web system. It contains the following information: the absolute address of the object repository, the name of the WOX repository, the list of WOX objects in the repository, the version of the system, the operating system on which the web system runs, and the frequency at which the garbage generated in *WOXRP* is deleted. Additionally, the *WOX Server* class includes the following methods for the operation of the web system: searching for an object,

adding an object to the repository, deleting an object from the repository, retrieving the class of a WOX object, obtaining the methods of a specific WOX object, and more.

The WOX Object class is used to represent each object stored in the object repository. When an object is uploaded to the repository, a *WOX Object* class instance is created by the web system. This instance stores the following information about the uploaded object: a unique ID representing the newly created object, the name of the XML file that contains the object, and the class to which the object belongs.

The WOX Visualizer is the class responsible for processing WOX objects. It extracts all the XML code to generate a visualization. The main functions of the program include the following: providing a format (indentation and color) to the labels of the WOX file to display them to the user; processing WOX objects and identifying their data type (primitive, lists, arrays, references, among others); generating tables that represent each object once its data type has been identified, using the *WOX Tab Generator* class.

WOX Tab Generator is the class that generates the graphic representation of an object. As a result, it generates an HTML table. With the help of its methods, content can be added to the table. This content can include attributes of the object it represents or buttons that provide access to the visualization of the object's methods.

WOX Invoker is the class that processes all the information regarding method invocation. Its main functions include the following: creating special fields displayed to the user for entering the required parameters for a method invocation; accepting objects used by the user as parameters for a method invocation, whether it is an existing object in a URL or an object uploaded to the server; handling the responses generated by the methods on invocation; and executing a specific method based on the user-entered parameters.

5 WEB SYSTEM IN OPERATION

This section presents the operational web system. It illustrates the storage of objects, the access to the repository, the visualization of a WOX object, the visualization of the methods of a WOX object, and the execution of a specific method.

5.1 Storage of objects

Figure 4 illustrates the operation of the web system for storing WOX objects, depicting a series of steps executed between the client and the server.

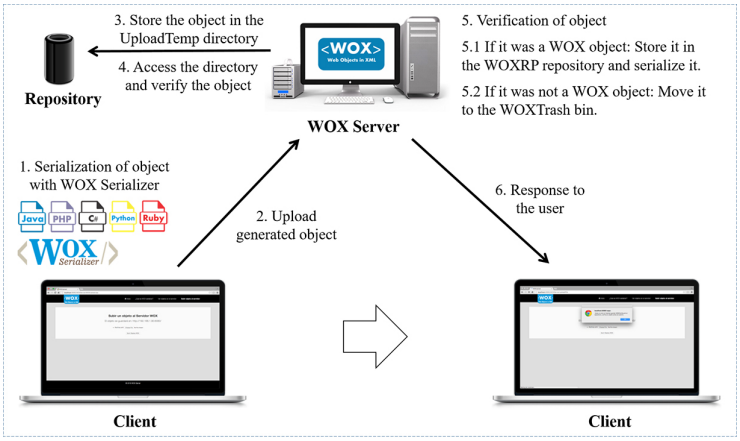

Fig. 4. Storing a WOX object

Step 1. The object can be serialized using the WOX serializer, which supports all programming languages compatible with WOX.

Step 2. The user proceeds to upload the object to the server by selecting the option *Upload an object to the server* from the main menu of the web system.

Step 3. The server stores the object in the *Upload Temp* directory of the repository.

Step 4. The system proceeds to verify the object by deserializing it to ensure that it is free of errors and does not already exist. Depending on the verification, the system executes either step 5.1 or step 5.2.

Step 5.1. If it was a WOX object, the web system stores it in the repository and serializes it.

Step 5.2. If it was not a WOX object, then the web system moves the object to the trash (*WOX Trash* bin).

Step 6. The user is informed about the result of storing the WOX object in the web system's repository.

It should be noted that for the registration of an object in the repository, the following actions are carried out: a new ID is generated to represent the new object; the class of the new object is obtained; the name of the file containing the new object is extracted; a *WOX Object* is created; the corresponding parameters are assigned to the object; and finally, it is added to the list of existing objects in the repository.

5.2 Access to the repository of objects

Figure 5 illustrates the operation of the repository for visualizing WOX objects, depicting a series of steps that occur between the client and server.

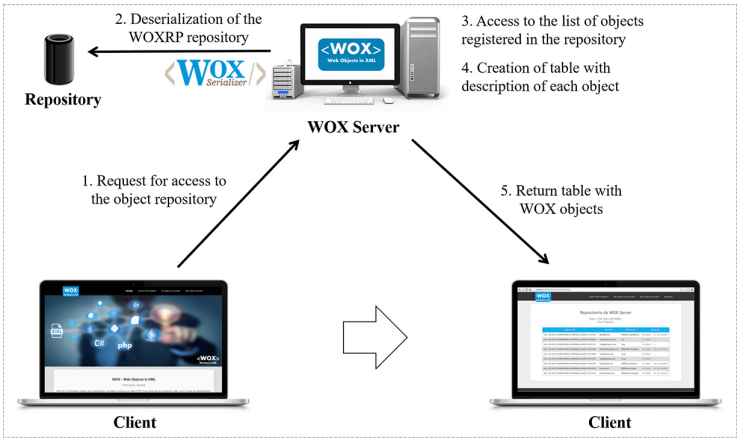

Fig. 5. Access to the repository for visualization of objects

Step 1. The user accesses the repository by selecting the option *View objects* in the server from the main menu of the web system.

Step 2. The *WOXRP.xml* object is deserialized, which is a list of objects of the *WOX Object* class that contains information about all the objects stored in the repository.

Step 3. Once the *WOXRP.xml* object has been deserialized, the information of the registered objects can be accessed. Subsequently, a table is generated to contain the URL and class of each registered object.

Step 4. Each registered object offers two options for the user: one to view the object and another to access its methods.

Step 5. The table displaying the objects is presented, allowing the user to select an object for visualization.

5.3 Visualization of a WOX object

Figure 6 illustrates the operation of the web system for visualizing a WOX object, depicting a series of steps executed between the client and server.

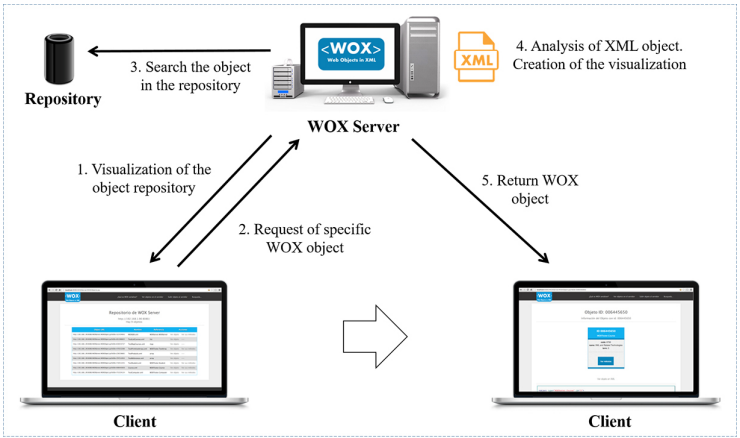

Fig. 6. Visualization of a WOX object

Step 1. The user accesses the repository of objects, where each 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 action's menu for a specific object. The user can also access the object directly using 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 achieve this, the *WOXRP.xml* object is deserialized, and the location of the file containing the requested object (if it exists) is accessed.

Step 4. Once the location and name of the file have been obtained, the file is analyzed using the *view Object BYXML* method. This method is responsible for creating an HTML table that represents the graphical display of the WOX object. It includes the object's id, the class it belongs to, and the attributes it contains. Finally, it returns all the generated HTML code and displays it to the user.

5.4 Visualization of the methods of a WOX object

Figure 7 illustrates the operation of the system for visualizing all the methods of a WOX object through a series of steps that occur out between the client and server.

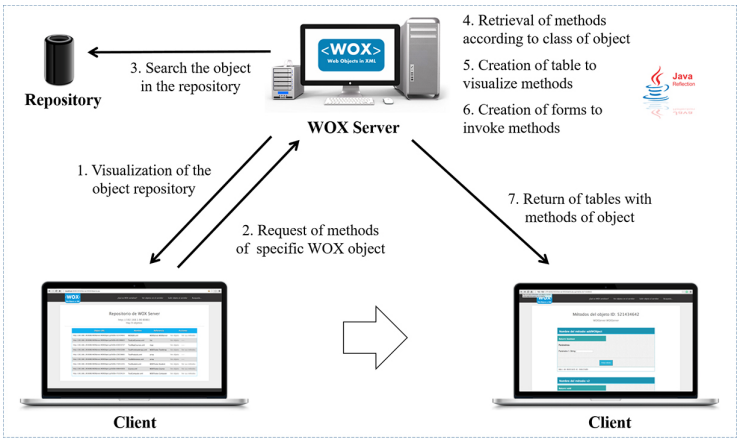

Fig. 7. Visualization of the methods of a WOX object

Step 1. The user accesses the repository of objects, where each 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 action's menu for a specific object in the repository.

Step 3. The system receives a request containing the id of an object to be manipulated and then proceeds to search for it in the object repository.

Step 4. If the object is found, the system retrieves the class of the object. From the class, the system retrieves all the methods and parameters required to invoke each method. In order to accomplish this, the object utilizes 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). The table will include the following attributes: method name, type of response it returns when invoked (object, int, float, etc.), button to invoke the method, and field where the response will be displayed.

Step 6. The system adds a form once the table representing each method has been created, in which the user provides 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.

5.5 Invocation of a method on a WOX object

Figure 8 illustrates the process of invoking a method on a specific WOX object, depicting a sequence of steps executed between the client and server.

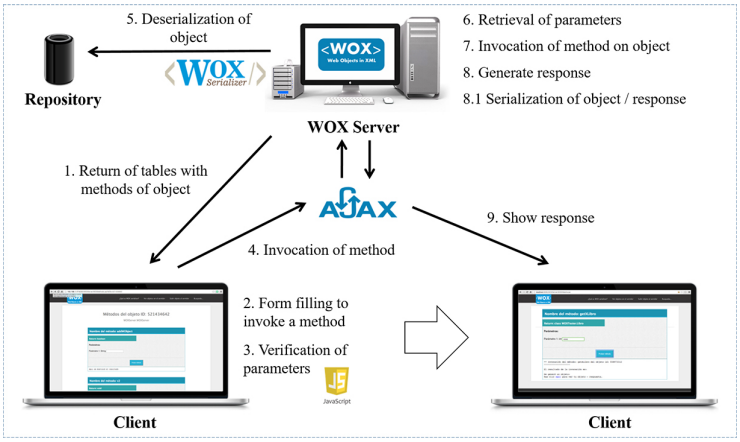

Fig. 8. Invocation of a method on a WOX object

Step 1. The user receives tables representing the invocation of methods on a specific object.

Step 2. The user proceeds to fill out the form (parameter entry) in order to invoke a specific method.

Step 3. While the filling out the form, Java script it is used to verify 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 indeed enters an integer, and not a decimal or text.

Step 4. After the user has entered the correct parameters to invoke a specific method, they can proceed to press the test method button. With the assistance of an asynchronous JavaScript call (AJAX), the response is displayed just below the table that represents the method to be executed. Without needing to access another tab

or refresh the browser page, AJAX sends a request to the server and forwards all the parameters entered by the user for the invocation of a specific method.

Step 5. The system receives the AJAX request with the following information: the id of the object containing the method to be invoked, the name of the method to be invoked, a list of parameters required to invoke the method, and a list of classes to which each parameter belongs. An example of the information that the server receives is the as follows: id: 3294242; method name: “add Book”; 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 based on the provided id, deserializes it, and verifies the list of parameters (ensuring 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 named “answer” is created to store the result of the method execution.

Step 8. The system proceeds to verify the answer object once the requested method has been invoked, obtaining its class. If the answer is a primitive object (int, float, char, 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 using the *WOX Serializer*.

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

5.6 Web system interface

This subsection presents some of the main interfaces of the web system in operation, based on the functionality described in previous sections.

Figure 9 shows a screenshot of the object repository. The user can access this interface by selecting *View objects in the server* from the main menu of the system.

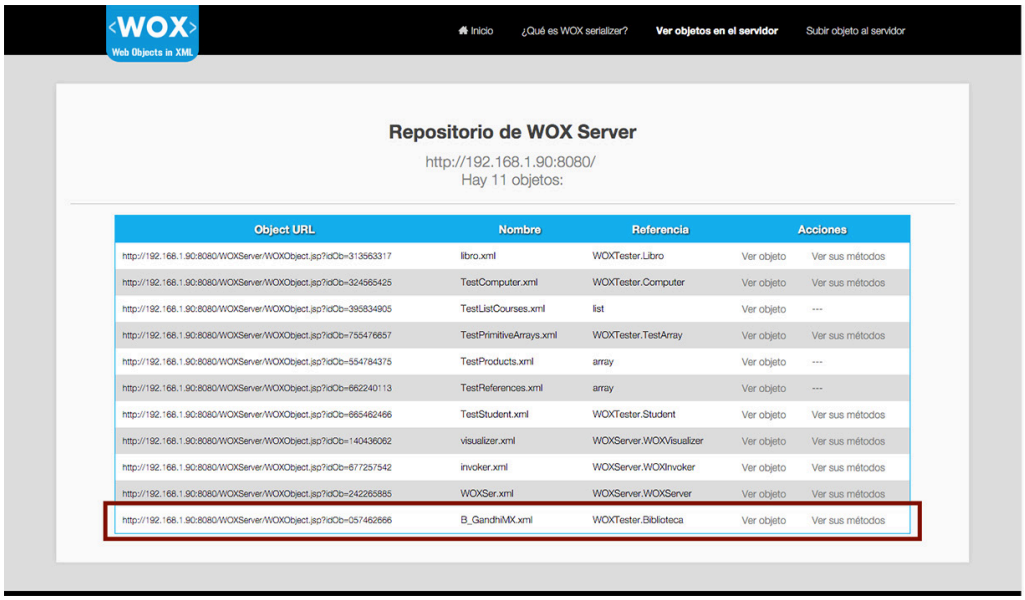

Fig. 9. Access to the repository for visualization of objects

The object repository has four columns: the URL of the object on the server, the name of the object, the remote reference to the object, and the actions that can be executed on the object (*View object* and *view its methods*). The *View Object* hyperlink

displays the graphical representation of the object and its XML representation, as illustrated in Figure 10.

Fig. 10. Visualization of an object graphically and in XML

In order to visualize the methods of an object, it is necessary that the object class be hosted on a WOX server. If this is the case, by clicking the *View methods* button in the object repository, the server will display to the user all the methods of that specific object, along with their corresponding fields to be invoked. This is illustrated in Figure 11, which shows the "*add Book*" method of the *library* class.

Fig. 11. Interface to visualize the *add Book* method of the *library* class

Figure 12 shows the *add Book* method, which adds a book to a *library* object, using 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 12 illustrates the results of the method invocation, the book has been added to the *library* object with the specified values.

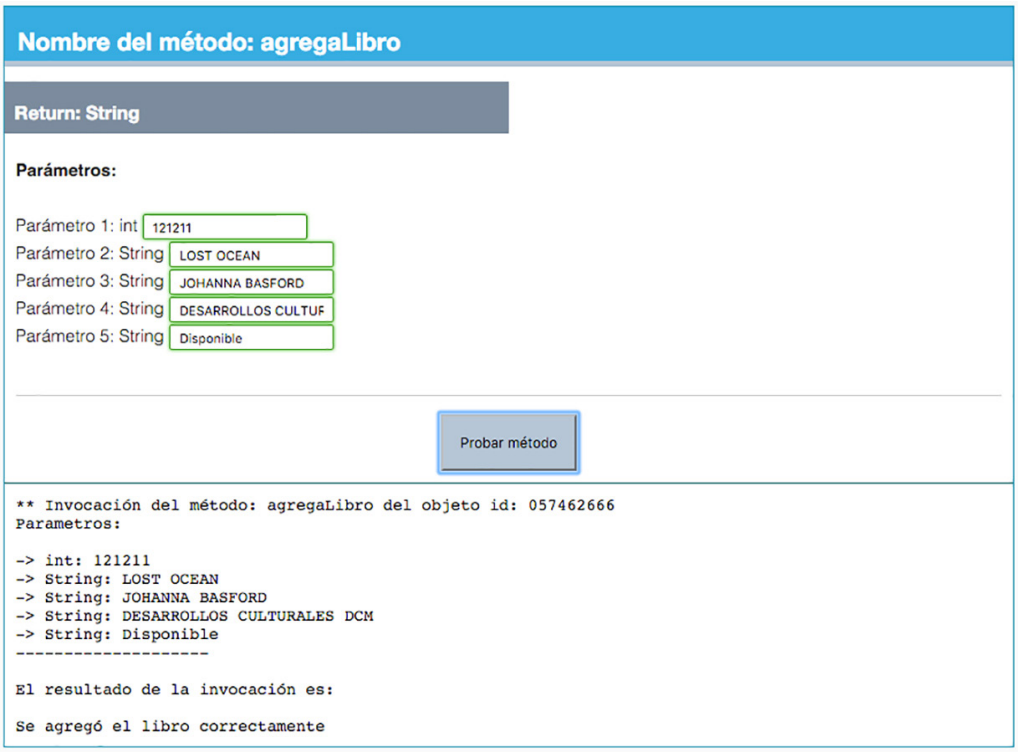

Fig. 12. Result of the invocation of the *add Book* method with the values specified

After adding two books to the *library* object through its methods, they can be displayed, as illustrated in Figure 13. It can be observed that the *library* object has changed, it now contains two books that were added previously.

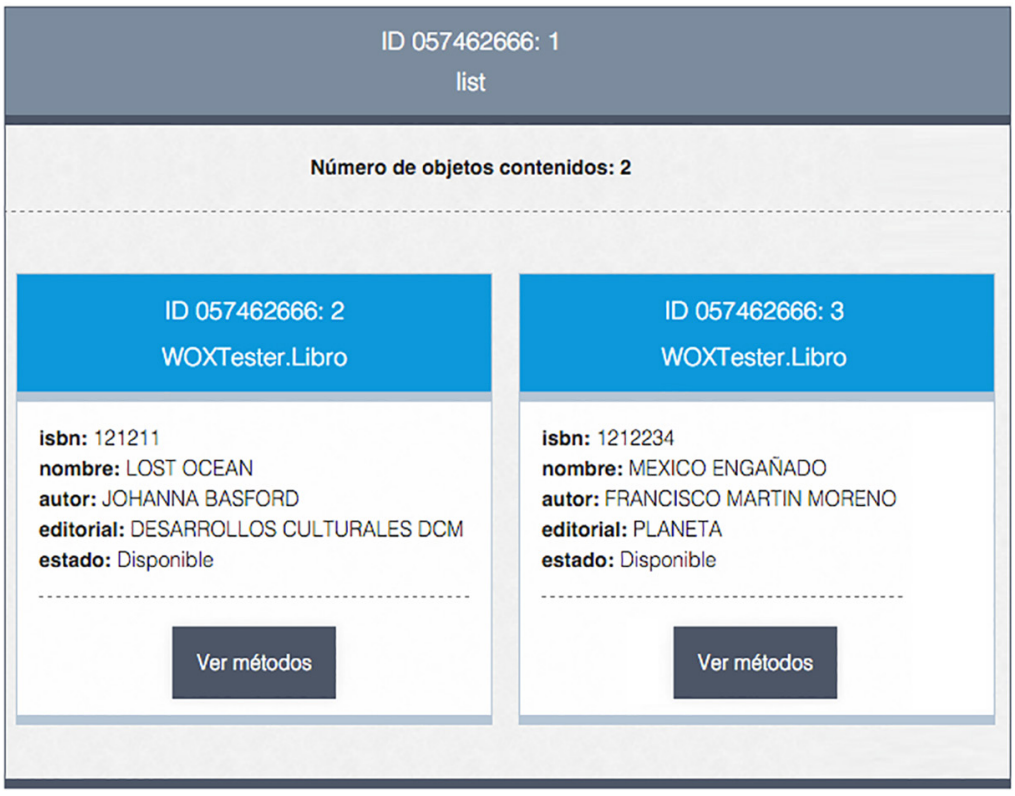

Fig. 13. The *library* object with two books added

6 CONCLUSIONS AND FUTURE WORK

This paper introduces a web system that was developed to support the teaching of an undergraduate distributed systems course. It describes a web system that was developed to complement the functionality of web objects in the XML (WOX) framework. The system allows users to store and visualize the state of distributed objects, display them, and execute methods through a web interface. Users can provide values for each method's parameters. The paper describes the functionality of the web system, showcasing its operation. Specifically, it illustrates the storage of objects, access to the repository, visualization of a WOX object, visualization of the methods of a WOX object, and the execution of a specific method.

A comparative analysis was conducted on five existing systems that share similarities with the developed web system. The most relevant features were highlighted, such as the interoperability among different programming languages, the use of XML as the object representation, the possibility to visualize objects and attributes, and the capability to execute methods on objects, among others.

Further work is needed to develop an evaluation tool to assess the web system with teachers and students of an undergraduate distributed systems course, focusing primarily on three aspects: functionality, design, and didactic features. The system is also planned to be deployed on a web server, enabling teachers and students to utilize it for supporting topics related to distributed objects and system interoperability.

7 REFERENCES

[1] C. R. Jaimez-González and S. M. Lucas, "Implementing a state-based application using web objects in XML," in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS. OTM 2007*, R. Meersman and Z. Tari, Eds., Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2007, vol. 4803, pp. 577–594. https://doi.org/10.1007/978-3-540-76848-7_40

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

[3] C. R. Jaimez-González, W. A. Luna-Ramírez, and 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*, 2012, pp. 127–132. https://ieeexplore.ieee.org/document/6470883

[4] C. R. Jaimez-González and S. M. Lucas, "Web Objects in XML (WOX): Efficient and easy XML serialization of Java and C# objects," 2018. http://woxserializer.sourceforge.net/

[5] C. R. Jaimez-González and A. Rodríguez, "Web Objects in XML (PyWOX): Object to XML serializer in the Python programming language," 2022. https://pywoxserializer.sourceforge.net/

[6] C. R. Jaimez-González and L. Hernández, "Web Objects in XML (PHPWOX): Object to XML serializer in the PHP programming language," 2022. http://phpwoxserializer.sourceforge.net/

[7] P. Merle, C. Gransart, and J. Geib, "CorbaWeb: A generic object navigator," 1996. http://www.lifl.fr/~merle/papers/96_WWW5/paper/Overview.html

[8] S. Chang and 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, and 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.

[10] G. Kumar and 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] M. Tork, M. Arya, L. Haas, M. Carey, W. Cody, R. Fagin, P. Schwarz, J. Thomas, and E. Wimmers, "The garlic project," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, New York, 1996.

[13] C. Burger and K. Rothermel, "A framework to support teaching in distributed systems," *Journal on Educational Resources in Computing*, vol. 1, no. 1, p. 3, 2001. https://doi.org/10.1145/376697.376698

[14] J. Wein, K. Kourtchikov, Y. Cheng, R. Gutierez, R. Khmelichek, M. Topol, and C. Sherman, "Virtualized games for teaching about distributed systems," *ACM SIGCSE Bulletin*, vol. 41, no. 1, pp. 246–250, 2009. https://doi.org/10.1145/1539024.1508955

[15] M. Arroyo, "Teaching parallel and distributed computing to undergraduate computer science students," in *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, Cambridge, MA, USA, 2013, pp. 1297–1303. https://doi.org/10.1109/IPDPSW.2013.276

[16] R. Fielding, "Architectural styles and design of network-based software architectures," PhD thesis, USA, 2000.

8 AUTHORS

Carlos R. Jaimez-González is a Professor at the Information Technology Department at the Universidad Autónoma Metropolitana Campus Cuajimalpa, in Mexico City. He received his PhD degree in Computer Science from the University of Essex, United Kingdom. His research interests include technologies for supporting education, interoperability in distributed systems, XML and related technologies, and the development of web and e-commerce applications. He has a distinction as a national researcher from the Mexican Government (E-mail: cjaimez@cua.uam.mx).

José M. Hernández-Salinas is a mobile and web application developer. He completed a BSc degree in Information Technologies and Systems from the Universidad Autónoma Metropolitana Campus Cuajimalpa. His research interests include technologies for supporting education, mobile and web application development (E-mail: 2123064604@cua.uam.mx).

Betzabet García-Mendoza is an Associate Professor at the Information Technology Department at the Universidad Autónoma Metropolitana Campus Cuajimalpa, in Mexico City. She received her MSc degree in Design, Information and Communication from the Universidad Autónoma Metropolitana. Her research interests include technologies for supporting education and web application development (E-mail: bgmendoza@cua.uam.mx).