Reci
Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

AsyncMonitor: una herramienta web para apoyar la enseñanza de la comunicación asíncrona en cursos de sistemas distribuidos

AsyncMonitor: A web tool to support teaching of asynchronous communication in distributed systems courses

AsyncMonitor: uma ferramenta web para apoiar o ensino de comunicação assíncrona em cursos de sistemas distribuídos

Carlos R. Jaimez-González
Universidad Autónoma Metropolitana, Unidad Cuajimalpa, México
cjaimez@correo.cua.uam.mx

Resumen

La comunicación asíncrona en aplicaciones cliente-servidor es muy útil para ejecutar en un servidor procesos que toman una cantidad considerable de tiempo, evitando que el cliente sea bloqueado en espera de que el proceso finalice su ejecución. Dada la importancia de la comunicación asíncrona en las aplicaciones distribuidas, en este artículo se presenta *AsyncMonitor*, una herramienta web que permite monitorear invocaciones de métodos asíncronos en el *framework Web Objects in XML* (WOX) y que, al mismo tiempo, apoya la enseñanza de los conceptos de la comunicación asíncrona en cursos de sistemas distribuidos.

AsyncMonitor permite al alumno visualizar gráficamente una cola de procesos con sus respectivos estatus (pendiente, activo y completado), además de visualizar la representación de los objetos que son el resultado de la ejecución de los métodos asíncronos. En el artículo también se describe la implementación de la comunicación asíncrona en WOX, la cual utiliza los mecanismos de sondeo y transmisión, con lo cual es posible que una aplicación ejecute no solo métodos síncronos sobre objetos remotos, sino también asíncronos.

Se presentan las pruebas que se realizaron con el objetivo verificar la funcionalidad de *AsyncMonitor* para diferentes tareas: recibir solicitudes de invocaciones de métodos asíncronos;

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

la correcta ejecución de los métodos con sus estatus correspondientes; la visualización de los objetos resultantes, así como la funcionalidad de los mecanismos de comunicación asíncrona de sondeo y transmisión. Las pruebas contemplan 20 escenarios: 10 utilizando el mecanismo de comunicación asíncrona de sondeo y 10 utilizando la técnica de transmisión. En cada escenario se mantuvo disponible un servidor WOX y cinco computadoras cliente, las cuales enviaron un número aleatorio de solicitudes de invocaciones de métodos asíncronos al servidor WOX. Los resultados obtenidos en las pruebas realizadas son alentadores, ya que 100% de las solicitudes enviadas por las computadoras cliente al servidor WOX fueron encoladas en la cola de métodos, lo cual significa que el mecanismo para recepción de solicitudes y encolamiento funciona correctamente. Con respecto a los métodos que fueron encolados, 85% de ellos terminaron su ejecución; cambiaron adecuadamente su estatus y fue posible visualizar el resultado de su ejecución en el navegador web.

Palabras clave: comunicación asíncrona, herramienta de enseñanza, herramienta web, sistemas distribuidos, *Web Objects in XML*.

Abstract

Asynchronous communication in client-server applications is very useful in order to execute in a server processes that take a considerable amount of time, avoiding that the client blocks waiting for the process to complete its execution. Given the importance of asynchronous communication in distributed applications, this paper presents *AsyncMonitor*, a web tool that allows to monitor asynchronous methods in the *Web Objects in XML* (WOX) framework, at the same time it supports teaching of asynchronous communication concepts in distributed systems courses.

AsyncMonitor allows students to visualize graphically a queue of processes with their corresponding status (pending, active and completed); it also visualizes the representation of objects, which are the result of the asynchronous methods execution. The paper also describes the implementation of asynchronous communication in WOX, which uses the polling and streaming mechanisms in order to allow an application to execute not only synchronous methods on remote objects, but also asynchronous methods.

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Tests were carried out with the aim of verifying the functionality of *AsyncMonitor* for different tasks: requests for asynchronous method invocations, the correct execution of methods with their corresponding status, the visualization of resulting objects, as well as the functionality of the polling and streaming asynchronous communication mechanisms. The tests have 20 scenarios: 10 using the polling mechanism and 10 using the streaming mechanism. In each scenario there is a WOX server available and five client computers, which send a random number of requests for asynchronous method invocations to the WOX server. The results from the tests carried out are encouraging, because 100% of the requests sent by client computers were queued in the methods queue, which means that the mechanism for reception of requests and queuing work correctly. Concerning the methods that were queued, 85% of them completed their execution; change their status appropriately; and it was possible to visualize the results of their execution on the web browser.

Keywords: asynchronous communication, teaching tool, web tool, distributed systems, Web Objects in XML.

Resumo

A comunicação assíncrona em aplicativos cliente-servidor é muito útil para executar em processos de servidor que levam uma quantidade considerável de tempo, evitando que o cliente seja bloqueado aguardando o processo para finalizar sua execução. Dada a importância da comunicação assíncrona em aplicações distribuídas, este artigo apresenta o AsyncMonitor, uma ferramenta web que permite monitorar invocações de métodos assíncronos no framework Web Objects em XML (WOX) e, ao mesmo tempo, suporta o ensino de os conceitos de comunicação assíncrona em cursos de sistemas distribuídos.

O AsyncMonitor permite que o aluno visualize graficamente uma fila de processo com seu respectivo status (pendente, ativo e completo), bem como visualize a representação dos objetos que são o resultado da execução dos métodos assíncronos. O artigo também descreve a implementação de comunicação assíncrona no WOX, que utiliza os mecanismos de polling e transmissão, com o qual é possível que um aplicativo execute não apenas métodos síncronos em objetos remotos, mas também assíncronos.

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Apresentamos os testes que foram realizados para verificar a funcionalidade do AsyncMonitor para diferentes tarefas: receber pedidos de invocações de métodos assíncronos; a execução correta dos métodos com seu status correspondente; a visualização dos objetos resultantes, bem como a funcionalidade dos mecanismos de polling e transmissão assíncronos. Os testes contemplam 20 cenários: 10 usando o mecanismo de comunicação de polling assíncrono e 10 usando a técnica de transmissão. Em cada cenário, um servidor WOX e cinco computadores clientes permaneceram disponíveis, o que enviou um número aleatório de pedidos de invocações de método assíncrono para o servidor WOX. Os resultados obtidos nos testes realizados são encorajadores, uma vez que 100% dos pedidos enviados pelos computadores clientes para o servidor WOX foram enfileirados na fila do método, o que significa que o mecanismo para receber pedidos e filas funciona corretamente. Com relação aos métodos colhidos, 85% deles terminaram sua execução; eles mudaram seu status corretamente e foi possível visualizar o resultado de sua execução no navegador da Web.

Palavras-chave: comunicação assíncrona, ferramenta de ensino, ferramenta web, sistemas distribuídos, Web Objects em XML.

Fecha Recepción: Noviembre 2016 **Fecha Aceptación:** Junio 2017

Introduction

Asynchronous communication in client-server applications is used for processes that need to be completed by a server and that take a considerable amount of time. If a client application runs a long process without using asynchronous communication, the client application will be blocked while the server is running the process; and it will be unlocked when the process finishes, that is, when the server sends the response back to the client application. Using asynchronous communication, a client program can continue working without blocking, while the process is being executed by the server.

Examples of applications that require the use of asynchronous communication are the following: a program that requests to execute a complex algorithm, which can take several minutes to complete; and the execution of a pattern recognition application, which can take a long time to perform the training and recognition processes.

In applications that operate on the Internet, the Hypertext Transfer Protocol (HTTP) (Fielding et al., 2007) is the most commonly used to communicate; however, it does not provide a way to open connections to clients and notify them when a process has finished. As part of the previous research that has been carried out, a framework was developed for programming objects distributed in the Java and C # programming languages, called Web Objects in XML (Web Objects in XML, WOX, for its acronym in English), which has been used to implement applications based on distributed objects that operate on the Internet, such as those presented in Jaimez-González and Lucas (2007). This framework uses HTTP as a transport protocol, and objects represent them in the extensible markup language (XML Markup Language, for its acronym in English) (W3C, 2016), and provides synchronous and asynchronous communication between clients and servers.

Given the importance of asynchronous communication in distributed applications, this article presents AsyncMonitor, a tool that allows to monitor the invocations of asynchronous methods that are executed in the WOX framework, at the same time that it supports the teaching of the concepts of communication Asynchronous in distributed systems courses, because it provides a graphical interface to visualize the processes that are running in its different states.

The rest of the article is organized as follows. The following section presents an introduction to the WOX framework. Subsequently, the two mechanisms for performing asynchronous communication are explained: HTTP polling and HTTP transmission. The implementations of asynchronous method invocations using polling and transmission are described in two subsequent sections. A following section is devoted to presenting AsyncMonitor, the web tool of teaching support to monitor invocations of asynchronous methods. The penultimate section presents the functionality tests that were performed and the results obtained. Finally, conclusions and future work are provided.

Web objects in XML

This section provides a brief introduction to the WOX framework, which combines features of distributed object-based systems and distributed web-based systems. Some of the characteristics of WOX are presented in the following paragraphs.

WOX uses Uniform Resource Locators (URL) to uniquely identify remote objects, following the principles of Representational State Transfer (REST) (Fielding, 2000). This feature is very important, since all objects are uniquely identified using their URL and can be accessed from anywhere on the web, either through a browser or through an application.

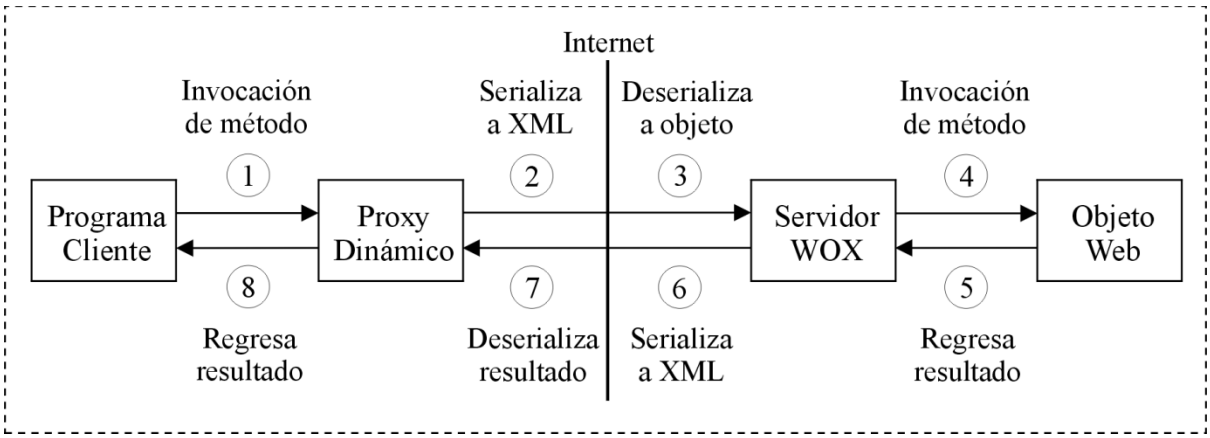
This framework uses an efficient and easy-to-use serializer called the WOX serializer (Jaimez-González and Lucas, 2011a, Jaimez-González, Lucas and López-Ornelas, 2011, WOX, 2009), which is the basis of the framework for serializing objects , requests and responses exchanged between clients and servers. The serializer is an independent library based on XML, which is capable of serializing Java and C # objects to XML, as well as taking the XML representation to generate the Java and C # objects again. One of its main characteristics is the generation of standard XML for objects, which is independent of the programming language and allows interoperability between different programming languages oriented to objects. Applications developed in Java and C # programming languages can interoperate through this framework. It should be noted that two additional serializers have also been developed: one in the Python programming language, called PyWOX (PyWOX, 2014), and another in the PHP programming language, called PHPWOX (Hernández-Piña y Jaimez-González, 2016; PHPWOX, 2014).

WOX has standard and special operations that can be applied on remote and local objects, among which the following are included: the request of remote references, invocations of static methods (calls of web services), invocations of instance methods, destruction of objects , request of copies of objects, duplication of objects, updating of objects, registration of objects and invocations of asynchronous methods. Some of these operations are described in Jaimez-González and Lucas (2007).

An example of the mechanism used by WOX in an invocation to a method of a remote object is illustrated in Figure 1, and the steps that are carried out are detailed below.

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Figura 1. Invocación de un método de un objeto remoto en WOX.



Fuente: elaboración propia.

1. The WOX client program invokes a method over a remote reference (the way in which the client invokes a method over a remote reference is exactly the same as it does over 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 in memory and executes the requested method.
5. The result of the invocation of the method is returned to the WOX server.
6. The WOX server serializes the result to XML and returns the actual result to the client or a reference to that result. The result is stored on the server in the event that a reference is returned.
7. The WOX dynamic proxy receives the result and deserializes it to the appropriate object, either real object or remote reference.
8. The WOX dynamic proxy returns the result to the client program.

From the point of view of the client program, the invocation of the method is performed and the result of return is obtained in a transparent manner. The WOX client libraries carry out the process of serializing the request and sending it to the WOX server, as well as receiving the result of the invocation of the method and deserializing it.

WOX also provides a web interface, which allows the inspection of objects through a web browser, the execution of methods on objects, and the visualization of objects with three different modes of operation: xml, html and image (Jaimez-González, 2014). It also has the characteristic of storing and navigating remote objects, which allows a client program to retrieve child objects from a given root object, through its XML representation (Hernández-Salinas and Jaimez-González, 2016).

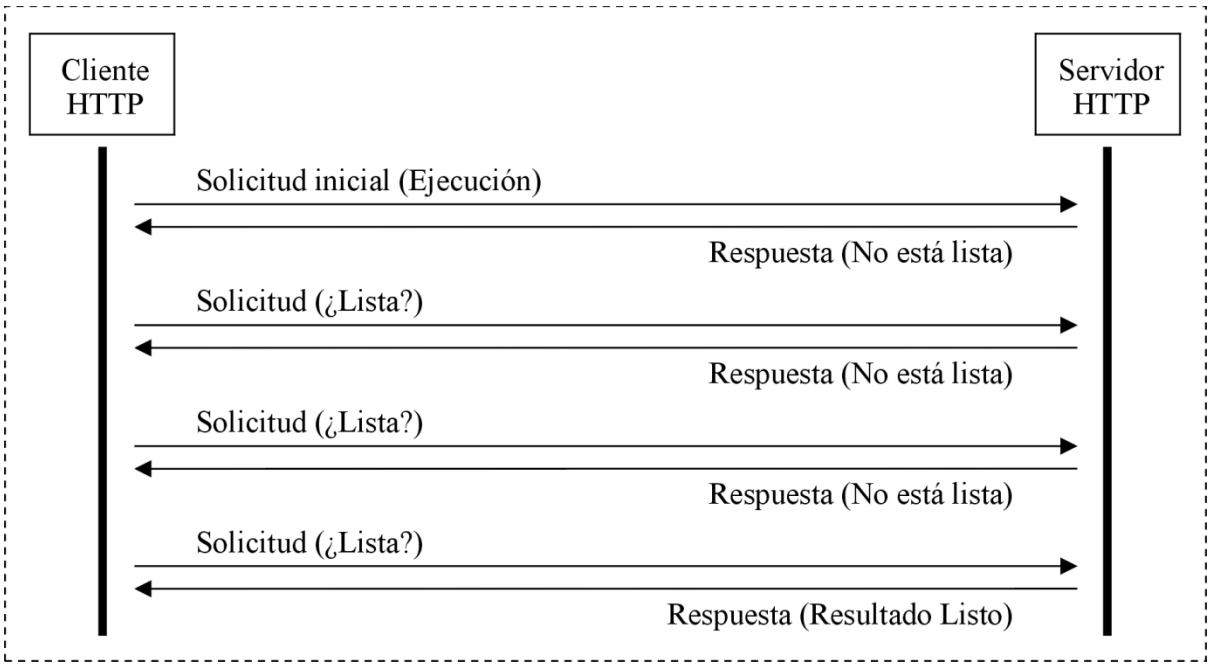
The following sections concentrate on the web tool to support the teaching of asynchronous communication, with which it is possible to monitor the invocations of asynchronous methods in WOX; as well as the implementation that was carried out of asynchronous communication using HTTP polling and transmission, based on the proposal of Jaimez-González and Lucas (2011b).

HTTP polling techniques and HTTP transmission

Figure 2 shows the HTTP polling technique, in which the client periodically makes requests to the server to obtain information about its original request. Each call is separated for a period of time. In this technique the first call of the client is really to make the original request of execution of the algorithm or program, and the subsequent calls are to poll if the original request has finished its execution. For each call the server will respond immediately either with the response of the completed process or with the indication that the process has not yet finished. This technique simulates a channel of communication in the background between the client and the server, and is very similar to the situation in which the server sends data directly to the client, as if the server really opened a connection to the client to notify him when the process is over.

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Figura 2. Comunicación asíncrona con técnica de sondeo HTTP.



Fuente: elaboración propia.

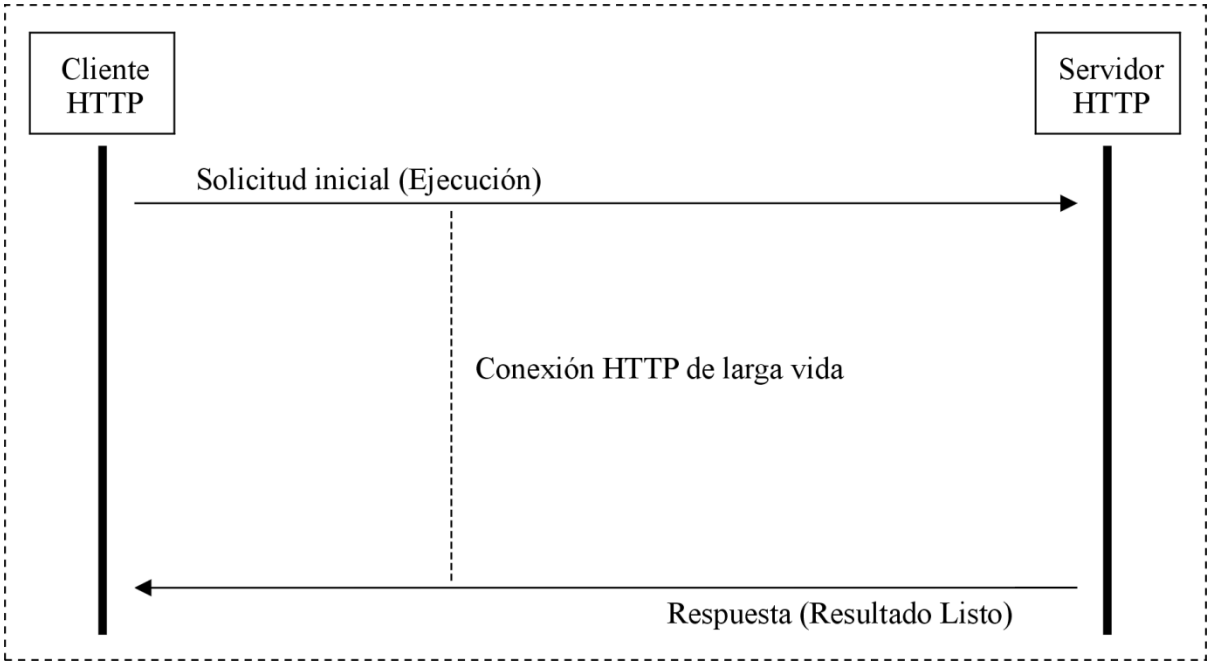
This approach has a significant cost due to the periodic calls to the server, since each call, no matter how small it may be, demands resources both on the client and on the server; each request also has an associated bandwidth cost. A key design of this approach would be to increase the average time of the period in which calls or surveys are made, while maintaining a good experience for the client; this depends on the type of application, but assuming that it is known that the algorithm or process can take a few minutes to complete, each customer call could be made every 30 seconds or a minute, for example; This will also depend on the requirements of the application and the number of users accessing it.

The HTTP transmission technique is shown in Figure 3, which shows that a long-lived HTTP connection is used. The client sends the request to the server, which keeps the connection open; When the process has finished its execution or there are updates that must be notified to the client application, the server sends the response or updates to the output channel. This approach has some disadvantages: long-life connections will inevitably fail, so there must be a recovery plan in case the connection breaks; Another disadvantage is that the servers can not maintain many open simultaneous connections, which are necessary if this solution is used. The

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

connection must be kept open for a reasonable period of time, and this will depend on the resources involved, such as the server, the network and the number of clients connected at a certain time, among other aspects.

Figura 3. Comunicación asíncrona con técnica de transmisión HTTP.



Fuente: elaboración propia.

Voelter, Kiercher y Zdun (2003) propose a design of asynchronous invocations in distributed object frameworks, which was also used for asynchronous invocations in web service framewors (Zdun, Voelter and Kircher, 2004). Bai and Tao (2010) propose a message intermediary using invocation of asynchronous methods with web services. In the case of the WOX framework, HTTP transmission and HTTP polling were implemented. Since WOX allows method invocations over remote web objects over HTTP, HTTP transmission was implemented by invoking asynchronous methods to allow a client application to remotely execute a method on an object that resides on a WOX server. When the method finishes its execution, the client application will be contacted and a specific action, defined by the client, could be executed.

In the case of HTTP polling, the implementation in WOX was also through invocations of asynchronous methods or processes that run on a WOX server. In this approach the process also runs on the server and when it is finished, the result is stored as a web object on the server, and exposed as any other object through its own URL. In this way, the client can retrieve the result of invocation of the asynchronous method as it does for any other object. Since an object in WOX is represented in XML, and can be accessed using any web browser, this approach is very convenient, since the result can be retrieved from anywhere on the Internet. For more information about the XML representation for Java and C # objects in WOX, see Jaimez-González, Lucas and López-Ornelas (2011), or visit the website of the Web Objects in XML project (WOX, 2009), from where it can be downloaded.

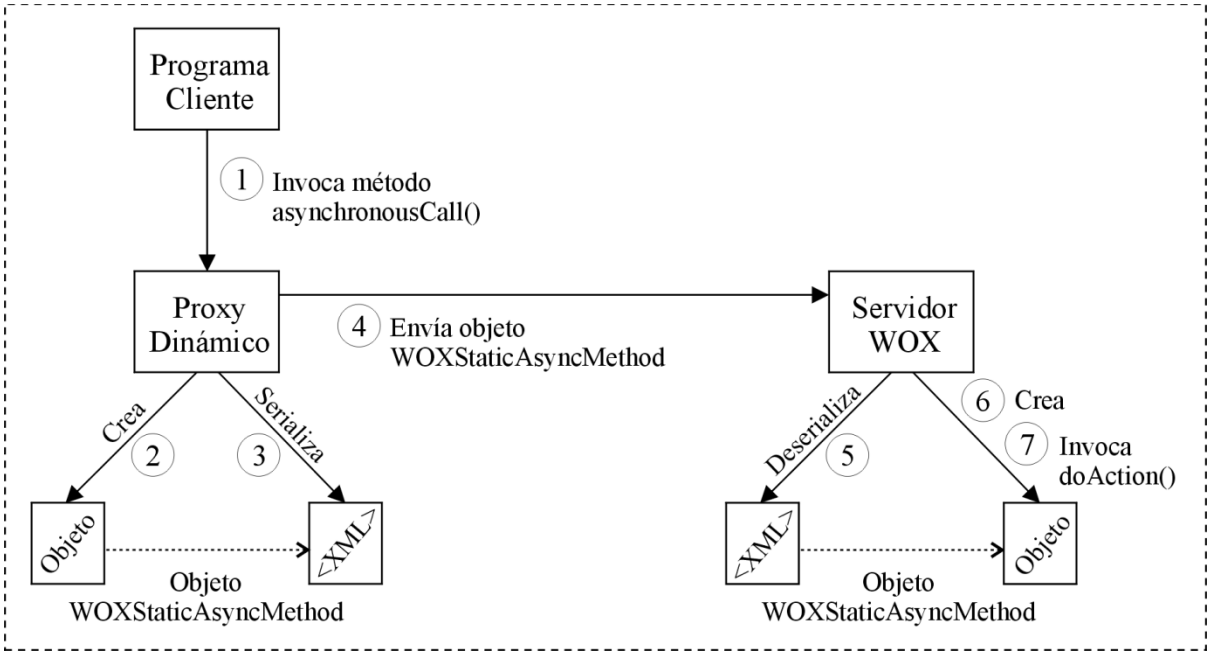
Asynchronous communication using polling

Asynchronous communication using the polling technique is carried out through invocations of asynchronous methods in the WOX framework. A client application invokes asynchronous methods on remote objects, which reside on a server. This section explains the process of invoking such methods from a client application, how they are executed on the server, and how they are implemented in WOX. It should be noted that the Java programming language is used for the code shown in this section.

Invocations of asynchronous methods are used when a method or process takes a considerable amount of time to complete; they prevent the client from crashing, waiting for the execution of the method on the server. Figures 4 and 5 show the steps carried out for an asynchronous method invocation in WOX using the polling technique, and an explanation of each step is provided below.

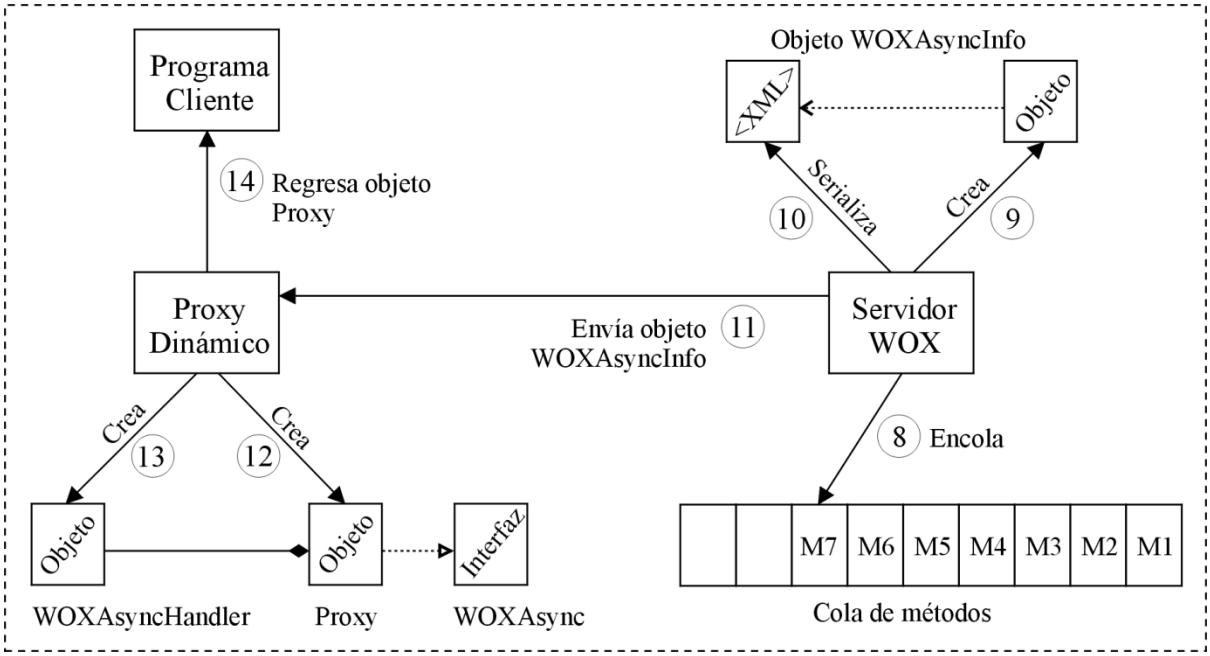
Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Figura 4. Invocación de método asíncrono utilizando técnica de sondeo.



Fuente: elaboración propia.

Figura 5. Invocación de método asíncrono y cola de métodos.



Fuente: elaboración propia.

1. In step 1 of figure 4, the client invokes a static method on a class that already resides on the server; it could also be an instance method of an existing object. This is done through the WOXProxy class by invoking the asynchronousCall () method, which has two variants, depending on the type of method, static or instance. The static variant, similar to a static method invocation, is shown below, in which it is observed that it takes as parameters: the URL of the server, the class where the method is located, the method that will be invoked, together with any parameter that be necessary. It should be noted that the return type of the asynchronousCall () method is a WOXAsync object.
2. WOXAsync woxAsync = (WOXAsync) WOXProxy.asynchronousCall (serverURL, className, methodName, params);
3. Step 2 of Figure 4 represents that the WOX client libraries transform the call of the asynchronousCall () method to a WOXStaticAsyncMethod object.
4. The WOX client libraries serialize the WOXStaticAsyncMethod object to XML, as shown in step 3 of Figure 4.
5. Step 4 of Figure 4 corresponds to the shipment of the serialized WOXStaticAsyncMethod object to the WOX server, in the same way as any other request from a client is sent.
6. The server receives the WOXStaticAsyncMethod object and deserializes it to make it an object, as shown in step 5 of Figure 4.
7. In step 6 of figure 4 the server is responsible for creating the object from the deserialization that was carried out in the previous step.
8. The server executes the doAction method of the object, as seen in step 7 of figure 4, in the same way as any other request is executed.
9. The execution of the doAction method will place the request of the asynchronous method in a queue for its later execution, where a thread is in charge of said queue of methods, as it is observed in step 8 of figure 5.
10. In step 9 of figure 5 the server will create a WOXAsyncInfo object, which contains enough information to track the execution of the method that is in the queue, such as the identifier of the process that was assigned to it.

11. The WOXAsyncInfo object is serialized to XML by the server, as seen in step 10 of Figure 5.
12. Step 11 of Figure 5 shows the shipment of the serialized WOXAsyncInfo object to the client, which will serve to monitor the execution of the method that is already in the queue.
13. The client program, through the WOX client libraries, receives the WOXAsyncInfo object and creates a Proxy object, as shown in step 12 of Figure 5, which implements the WOXAsync interface. This interface contains methods to monitor the execution of the asynchronous methods and processes that are in the execution queue.
14. In step 13 of Figure 5 the server creates a WOXAsyncHandler object to handle all method invocations on the Proxy object. Each call about the proxy instance will be handled by this object, which will contact the server to gather information about the method. The Proxy object is composed of a WOXAsyncHandler object, as shown in Figure 5.
15. The proxy object is returned to the client, as shown in step 14 of figure 5, with which you can monitor the execution of the asynchronous method found in the server's method queue.

Once the asynchronousCall method has returned, the client will be able to monitor the asynchronous method using the methods provided by the WOXAsync interface. The client can invoke the following methods on the woxAsync proxy:

getProcessId (). This method returns the processId process identifier, given by the server to the method. This information can be obtained from the WOXAsyncInfo object, which is sent immediately to the client.

getStatus (). This method returns an integer representing the status of the invocation of the method: Pending, which indicates that the method is still in the queue waiting to be executed; active (Active), which indicates that the method is being executed at this time; or completed (Completed), which indicates that the method has ended and the result is ready for it to be recovered.

getResultAsReference (). This method can be executed only when the getStatus () method has returned Completed, which means that the result of the execution of the method is ready for it to be retrieved. This method will obtain a reference to the result of the execution of the method.

getResultAsCopy (). This method is similar to the getResultAsReference () method, but the getResultAsCopy () method does not retrieve a reference to the result of the execution of the method, but a copy of the result.

For the methods that were mentioned, except for the getProcessId () method, the client will send the request to the server as a WOXMonitor object. The server will receive this request and process it in the same way as any other request from a client; that is, the server will execute the doAction () method of the WOXMonitor object, which will use the processId to identify the process and return an appropriate response to the client.

Each request that a client makes to invoke an asynchronous method is received by the server and glued; there is a thread that executes the methods in the queue sequentially. When a method has finished its execution, its result is stored as any other object in the WOX framework, and a reference to it is stored in the queue, which allows the server to provide a reference to the object when the client requests it. The following information is stored for each asynchronous method request: processId; method to be executed, object WOXMethod, static or instance method; status of the execution of the method (pending, active or completed); and a reference to the result of the execution of the method, by means of a WOXReference object.

As future work in this implementation, it is necessary to include a processURL attribute to the information stored in each asynchronous method request, which would represent a unique URL where each process would be monitored. You could also include the completed percentage of the process to know the progress you have, since at this time only the status of the process is reported: pending, active or completed.

Asynchronous communication using transmission

This section presents the other implementation of asynchronous communication, which uses the transmission technique, where the WOX client libraries are in charge of verifying if the process has finished, it is not the client program that does it, in such a way that when the process it ends its execution on the server, the client is notified and an action can be executed. The Java

programming language is also used for the code shown in this section. The action to be executed is specified by a method in the client program, and client libraries provide the asynchronousCall method for this purpose, as shown below.

WOXProxy.asynchronousCall (serverURL, className, methodName, params, objForCallback, callbackMethod);

The parameters of the asynchronousCall method are the following: serverURL is the URL where the server is located; className is the class where the method that will be invoked is; methodName is the name of the method that will be invoked on the server; params represents the parameters of the method that will be invoked; objForCallbak is a local reference of the object that has the callbackMethod method that will be invoked when the asynchronous method on the server finishes its execution.

In the implementation of the HTTP transmission technique that was performed, the client application invokes the asynchronousCall method previously shown, and can continue working without blocking. In this approach, the implementation of the asynchronousCall method is executed in a background process, which monitors the end of the process that is running on the server. The client application does not realize that there is a process running in the background. The process that is responsible for monitoring the server is very similar to the one previously described in the polling technique, but with the transmission technique, the WOX client libraries implement this functionality in the background.

When the process has finished running on the server, the process that is in the background monitoring its completion, requests the result and retrieves it; with the polling technique, this was carried out explicitly by the client program. The process running in the background invokes the method on the local object that was specified by the client program, that is, the callbackMethod method of the objectForCallback object; which is another difference with respect to the implementation of the polling technique, where the client program only retrieves the result. In the implementation of the transmission technique, the client libraries retrieve the result, which is available to the client program, but a method is also executed on a local object.

In summary, in the implementation of the polling technique, the client program is responsible for monitoring the asynchronous process until it finishes, and when the process has finished its execution, the client program can request a reference or a copy of the execution result

of the method, which is carried out by the WOXAsync interface. In the implementation of the transmission technique, a background process, which is part of the WOX client libraries, is responsible for monitoring the asynchronous process until it ends, so that when it finishes running on the server, the client program is notified, and an action defined by the client program can be executed.

AsyncMonitor

The WOX framework includes a web tool, called AsyncMonitor, for asynchronous methods and processes, which allows clients to monitor the queue of processes that are being executed on the server, and in turn supports the teaching of asynchronous communication in courses of distributed systems, since it graphically shows how asynchronous processes are executed in a queue and allows to visualize the status of each of them.

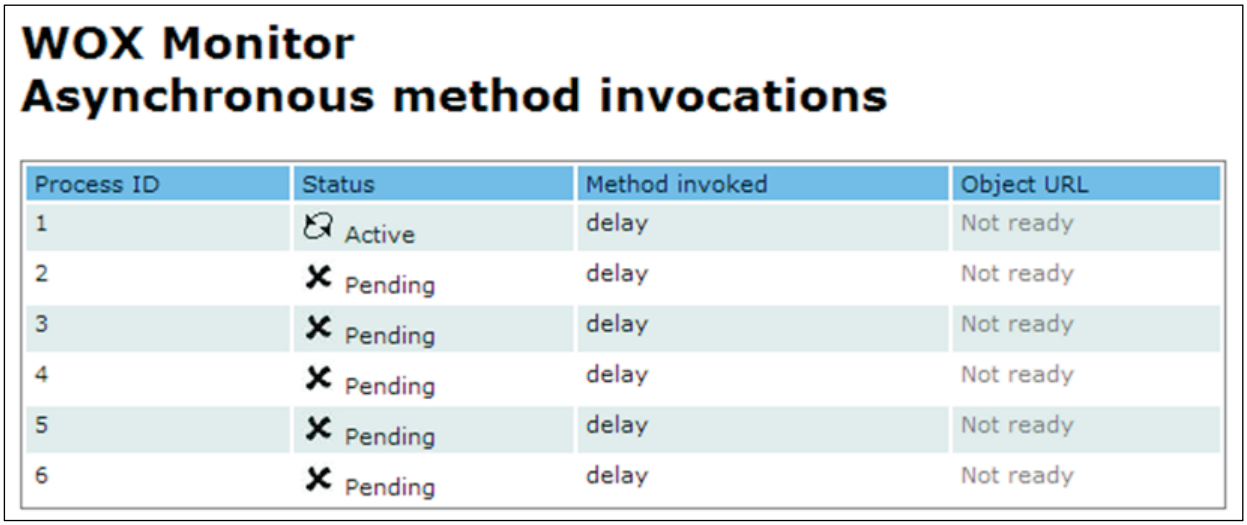
AsyncMonitor can be accessed through a web browser, in which it is possible to view information about the methods and processes that are currently being executed, since it provides the following: the process identifier; the status of the process (pending, active or completed); the name of the process; and the URL of the result of the process, which is actually an object, only when the method has finished its execution. This tool can be used to track invocations of asynchronous methods received in a particular WOX server and can be consulted in the URL shown below, where localhost must be replaced by the IP address or by the name of the computer where the server was installed. WOX:

http://localhost:8080/WOXServer/AsyncMonitor.jsp

Figure 6 shows a screenshot of AsyncMonitor, after having received six client requests that want to execute asynchronous methods. The status of the first process is active (Active), that is, this process is being executed by the server; the rest of the processes have the status of pending (Pending), which means that they are pending execution as observed by the URLs of the objects that are not ready, since all these methods have not finished their execution.

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Figura 6. *AsyncMonitor* con un proceso activo y el resto pendientes de ejecución.

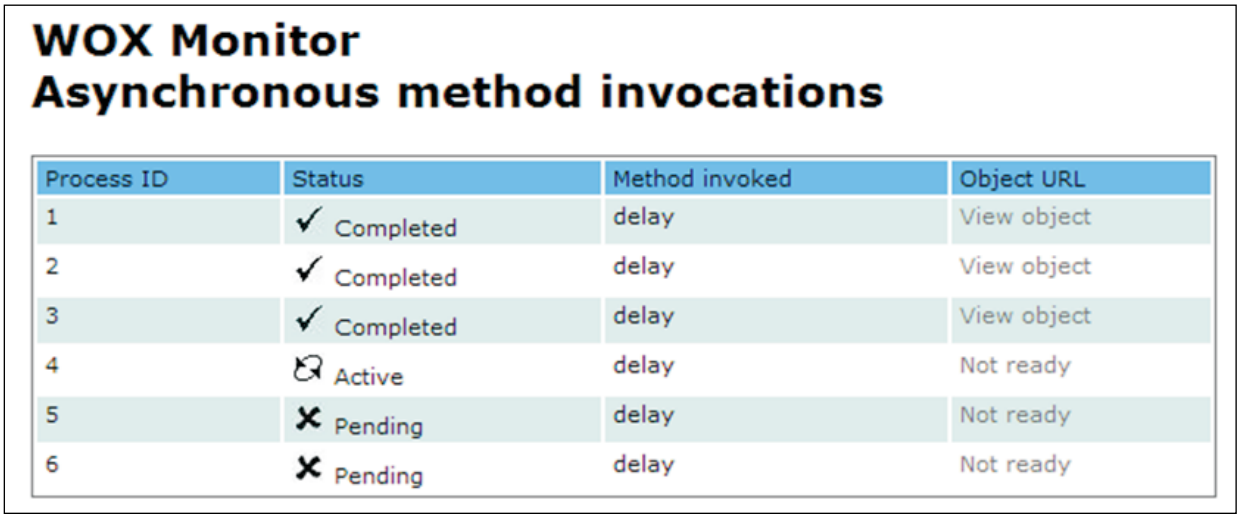


Fuente: elaboración propia.

Figure 7 shows a screenshot of AsyncMonitor a period of time after the one shown in Figure 6. The first three processes or method invocations have finished their execution. The results of the executions of these methods are ready and can be accessed through the hyperlink View object, which is a URL that leads to the result of the invocation of the method. The results can also be retrieved through a client program with the getResultAsCopy () or getResultAsReference () methods, which were presented in a previous section. The fourth method shown in figure 7 is still active, and the rest of the methods are pending execution.

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Figura 7. *AsyncMonitor* con procesos terminados, activos y pendientes de ejecución.



Fuente: elaboración propia.

AsyncMonitor is a very useful web tool to visualize all invocations of asynchronous methods that are being executed in a WOX server. It should be noted that a client program could also monitor invocations of asynchronous methods programmatically, using the methods previously presented.

The method queue in AsyncMonitor is implemented as a map in memory, since in this way it is easy and quick to access and the results of method invocations are stored as objects. There is the possibility of storing the map in a database or in some other storage medium each time the map is modified, which would happen in one of the following scenarios:

1. The arrival of a new request for the execution of an asynchronous method, since a new record is required on the map.
2. The beginning of the execution of a method, since the status of the process changes from slope to active.
3. The termination of a method, since the result is stored as an object and a reference to the result of the execution is added to the map.

The map would remain updated; If the server shuts down at any time, the map can be recovered when the server starts again, so that the server would need to restart with the process that has been marked as active.

The execution of the methods in the queue was implemented with a thread that handles the processes, which executes them sequentially in the order in which they are in the queue, that is, the order in which they have arrived. The thread goes to sleep when there are no more processes that must be executed in the queue, and wakes up when a new process arrives to be executed.

There are other approaches to implement the execution of methods in the queue; for example, a fixed number of threads could be contemplated, which will take care of the processes in the queue, where each thread could be created as needed. Another approach would be to create a thread for each asynchronous method invocation request that is received, so that each request is answered as soon as it arrives; however, the disadvantage is that you would have to create as many threads as concurrent requests would have, in addition to the processing resources that should exist in the server that would handle these requests.

An important situation that must be considered in the execution of methods in the queue is its priority, which could be assigned by the client application at the time of the execution request. The definition of policies for this type of problems is part of the future work that will be done for this support tool.

Tests and results

In this section the tests that were performed and the results obtained are presented. AsyncMonitor functionality tests were carried out, with the support of students and these were with different scenarios that are explained below.

The objective of the tests was to verify the functionality of AsyncMonitor for different tasks: to receive requests for invocations of asynchronous methods, the correct execution of the methods with their corresponding status, the visualization of the resulting objects, as well as the functionality of the execution threads using the two asynchronous communication mechanisms (HTTP polling and HTTP transmission).

Six computers were used for the tests. The WOX server was installed on a computer with the following features: Windows 10 Professional 64-bit operating system, Intel Core i7-7700K processor at 4.50 GHz, 16 GB RAM. The five computers that were used as clients had the following features: Windows 7 Professional 64-bit operating system, Intel Core i7-2760 processor at 2.40 GHz, 8 GB RAM.

A total of 10 test scenarios were performed for each asynchronous communication mechanism, that is, a total of 20 scenarios: 10 using HTTP polling and 10 using HTTP transmission. In each scenario the WOX server was kept available and the five client computers started to send a random number of asynchronous method invocation requests to the WOX server. On each client computer there was a program running in the Java language that made invocations of asynchronous methods on objects that resided in the WOX server, so that to send the method invocation requests, it used the WOX client libraries, which They were in charge of directly contacting the WOX server through a Proxy. On each client computer AsyncMonitor was accessed through a web browser to verify the information of their requests.

Table 1 shows the results of the first scenario using the HTTP polling mechanism, which were obtained through the interaction of the five client computers with the WOX server. Each row of the table represents the results of the invocations of a particular client computer (Client1, Client2, Client3, Client4 and Client5). With respect to the columns, the first of them indicates the number of method invocation requests sent by that client computer to the server; the second column contains the number of those requests that were pasted into the method queue that is on the server; the third column represents the number of applications that successfully completed its execution; the fourth column indicates how many of the requests sent correctly changed their status (pending, active and completed); and, finally, the fifth column shows for how many of the requests it was possible to visualize the result of its execution as an object.

In the first line of table 1, it can be seen that Client1 sent five asynchronous method invocation requests; five requests were pasted on the server; only three methods finished their execution, because two of them resulted in program exceptions, so they did not finish their execution; three applications correctly changed their status from pending to active and from active to completed (the two requests that resulted in the exception erroneously stayed with the status of active); the results of the execution of the three methods that ended correctly were visualized in the web browser in its XML representation. Similarly, the results of Client 2, 3, 4 and 5 can be read. For each of the 10 scenarios, a table was generated as shown in Table 1, where the number of requests that were randomly specified was randomly specified. each client computer sent to the WOX server, and the information was captured.

Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Tabla 1. Resultados del primer escenario con el mecanismo de sondeo HTTP.

	# solicitudes enviadas	# solicitudes encoladas	# solicitudes terminadas	Cambio correcto de estatus	Visualización de objetos
Cliente1	5	5	3	3	3
Cliente2	3	3	1	1	1
Cliente3	4	4	4	4	4
Cliente4	2	2	2	2	2
Cliente5	6	6	5	5	5

Fuente: elaboración propia con los resultados de las pruebas con *AsyncMonitor*.

Table 2 shows the concentration of the 10 scenarios that were made using the HTTP polling technique. It should be noted that the number of applications that were not completed is due to the fact that the methods did not finish their execution due to causes related to the code of the method, all of them triggered exceptions that are specific to the operation of the methods, without this having any relation with the correct functioning of AsyncMonitor. However, the change of status of those methods that did not finish successfully is a problem that must be corrected, since until the moment when a method does not finish its execution, its status remains established in active, giving the impression that the method is still running, when in fact it has already finished.

Tabla 2. Resultados de todos los escenarios con el mecanismo de sondeo HTTP.

	# solicitudes enviadas	# solicitudes encoladas	# solicitudes terminadas	Cambio correcto de estatus	Visualización de objetos
Cliente1	85	85	79	79	79
Cliente2	67	67	55	55	55
Cliente3	73	73	65	65	65
Cliente4	46	46	40	40	40
Cliente5	58	58	51	51	51

Fuente: elaboración propia con los resultados de las pruebas con *AsyncMonitor*.

Reci
Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

Similar to the results shown in Table 1, Table 3 shows the results of the first scenario now using the HTTP transmission mechanism. In the first row of the table, it can be seen that Client1 sent seven asynchronous method invocation requests; seven requests were pasted on the server; the seven methods finished their execution; the seven applications correctly changed their status from pending to active and from active to completed; the results of the execution of the seven methods that ended correctly were visualized in the web browser in its XML representation. Similarly, the results of Client 2, 3, 4 and 5 can be read. For each scenario, a table was generated, such as the one shown, where the number of requests that each client computer sent to the server was randomly specified.

Tabla 3. Resultados del primer escenario con el mecanismo de transmisión HTTP.

	# solicitudes enviadas	# solicitudes encoladas	# solicitudes terminadas	Cambio correcto de estatus	Visualización de objetos
Cliente1	7	7	7	7	7
Cliente2	7	7	5	5	5
Cliente3	3	3	3	3	3
Cliente4	6	6	4	4	4
Cliente5	5	5	3	3	3

Fuente: elaboración propia con los resultados de las pruebas con *AsyncMonitor*.

Table 4 shows the concentration of the 10 scenarios that were made using the HTTP transmission technique. It should be noted that the number of requests that were not completed (third column) is due to the fact that the methods that were executed did not finish their execution, similar to the results shown in table 2; in all cases this was due to the fact that exceptions were triggered by the operation of the particular methods, without this having any relation with the correct functioning of AsyncMonitor. Regarding the change of status of the applications, the same thing happens as in Table 2, where for those methods that did not finish their execution, their status remained erroneously active; the latter is something that should be corrected for the next version of AsyncMonitor.

Tabla 4. Resultados de todos los escenarios con el mecanismo de transmisión HTTP.

	# solicitudes enviadas	# solicitudes encoladas	# solicitudes terminadas	Cambio correcto de estatus	Visualización de objetos
Cliente1	77	77	68	68	68
Cliente2	91	91	83	83	83
Cliente3	80	80	62	62	62
Cliente4	53	53	45	45	45
Cliente5	68	68	60	60	60

Fuente: elaboración propia con los resultados de las pruebas con *AsyncMonitor*.

The results obtained in these functionality tests are encouraging, since they allow observing that there is a correct functioning of the execution of asynchronous methods with AsyncMonitor of the WOX framework. Of the total requests sent for invocation of asynchronous methods, 100% were queued in the server method queue. Although not all the requests finished their execution, this was not due to any problem with the WOX server, but because of the code itself of the methods that were sent to execute. The status of the methods that ended successfully successfully changed, while those of the methods that did not finish remained active, which indicates that work must be done at this point so that their status changes appropriately. The visualization of objects was also carried out successfully for those methods that finished their execution, showing the objects in XML representation.

Conclusions and future work

In this article, an introduction to a framework for programming objects distributed in Java and C # was presented, called Web Objects in XML (WOX), which has been used to implement distributed applications based on objects. WOX uses HTTP as a transport protocol, XML for the representation of objects, and provides synchronous and asynchronous communication between clients and servers.

The implementation of asynchronous communication in WOX was described, in which two different mechanisms were used: HTTP polling and HTTP transmission. Although there is no way in HTTP to open connections to clients to notify them when a process has finished,

asynchronous communication was simulated in WOX, using polling and transmission techniques; with which it was possible to have a robust and very useful implementation, in such a way that not only invocations of synchronous methods on remote objects are supported in WOX, but also invocations of asynchronous methods.

A web tool to monitor asynchronous processes in WOX was also presented, called AsyncMonitor, which is also considered as a support tool for the teaching of asynchronous communication in distributed systems courses, since it allows the student to graphically visualize a queue of processes with their respective status (pending, active and completed), in addition to visualizing the representation of the objects that are the result of the execution of the asynchronous methods.

Tests were carried out in order to verify the functionality of AsyncMonitor for different tasks: to receive requests for invocations of asynchronous methods, the correct execution of the methods with their corresponding status, the visualization of the resulting objects, as well as the functionality of the execution threads using the two asynchronous communication mechanisms (HTTP polling and HTTP transmission). The tests contemplated 20 scenarios: 10 using the HTTP polling asynchronous communication mechanism and 10 using the HTTP transmission technique. In each scenario, a WOX server and five client computers remained available, which sent a random number of requests for asynchronous method invocations to the WOX server. On each client computer, AsyncMonitor was accessed to verify the information of their requests.

The results obtained in the tests carried out are encouraging, since 100% of the requests sent by the client computers to the WOX server were queued in the queue of methods, which means that the mechanism for receiving requests and queuing works correctly. With respect to the execution of the methods that were glued, 85% finished their execution; the rest of the methods that did not finish their execution (15%) was due to exceptions that occurred due to the implementation of the same method that was sent as an application, but it was not due to the operation of AsyncMonitor or the WOX famework. With respect to the status of the glued methods, those methods that successfully completed their execution changed properly, that is, they changed from slope to asset and from asset to completion. In the case of the status of the methods that did not finish their execution, they remained active, which indicates that there is a problem that must be addressed for the next version of AsyncMonitor. The visualization of the

XML representation of the objects was satisfactory, for those methods that finished their execution successfully, since for the others there could not be visualization given that there is not a result to show, because the method did not finish its execution.

As future work are several activities, such as developing more utilitarian methods to measure the percentage of progress of a particular process, the time taken by a process so far, the user who is executing a specific process, among others. It is also necessary to create a policy to determine the priority of execution of asynchronous processes in AsyncMonitor, as well as to propose other approaches for execution of asynchronous methods and thread management.

Finally, it is necessary to perform concurrency tests of execution of asynchronous methods, in addition to the tests that have already been carried out. It also contemplates the realization of an evaluation instrument to measure the perception of the students regarding the support that is had with this tool in the teaching of asynchronous communication in the course of distributed systems.

Bibliography

Bai, F., Tao, W. (2010). Message Broker Using Asynchronous Method Invocation in Web Service and Its Evaluation. Proceedings of the Third International Conference on Software Testing, Verification, and Validation Workshops, París, Francia, pp. 265-273.

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (Tesis doctoral, University of California, Irvine). Recuperada de http://www.ics.uci.edu/fielding/pubs/dissertation/top.htm.

Fielding, R., Gettys, J., Mogul, J., Frysyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (2007). *The Hypertext Transfer Protocol, HTTP/1.1*. W3C. Recuperado de https://www.w3.org/Protocols/HTTP/1.1/rfc2616bis/draft-lafon-rfc2616bis-03.html.

Hernández-Piña, L., Jaimez-González, C. R. (2016). Serialización de Objetos PHP a XML. Revista Research in Computing Science, 125, pp. 87-95.

Hernández-Salinas, J. M., Jaimez-González, C. R. (2016). Herramienta Web para Almacenar y Visualizar Objetos Distribuidos. Revista Research in Computing Science, 125, pp. 63-74.

Jaimez-González, C. R. (2014). A Simple Web Interface for Inspecting, Navigating, and Invoking Methods on Java and C# Objects. Revista Research in Computing Science: Advances in Computing Science, 81, pp. 133-142.

Jaimez-González, C., Lucas, S. M. (2007). Implementing a State-based Application Using Web Objects in XML. En R. Meersman y Z. Tari (Eds.), *Lecture Notes in Computer Science*, Vol. 4803/2007 (pp. 577-594). Berlin, Alemania: Springer-Verlag.

Jaimez-González, C. R., Lucas, S. M. (2011a). Interoperability of Java and C# with Web Objects in XML. Proceedings of the International Conference e-Society (ES 2011), Ávila, España, pp. 518-522.

Jaimez-González, C. R., Lucas, S. M. (2011b). Asynchronous Method Invocations Using HTTP Polling and HTTP Streaming. Proceedings of the International Conference on Applied Computing 2011 (AC 2011), Río de Janeiro, Brasil, pp. 536-540.

Jaimez-González, C., Lucas, S., López-Ornelas, E. (2011). Easy XML Serialization of C# and Java Objects. *Proceedings of the Balisage: The Markup Conference 2011*, USA, Vol. 7. doi:10.4242/BalisageVol7.Jaimez01.

Voelter, M., Kircher, M., Zdun, U. (2003). Patterns for Asynchronous Invocations in Distributed Object Frameworks. *Proceedings of EuroPlop 2003*, Irsee, Alemania.

Web Objects in XML (WOX) [Software] (2009). Essex, Reino Unido: University of Essex. Recuperado de http://woxserializer.sourceforge.net/.

Web Objects in XML in PHP (PHPWOX) [Software] (2014). México: Universidad Autónoma Metropolitana. Recuperado de http://phpwoxserializer.sourceforge.net/.

Web Objects in XML in Python (PyWOX) [Software] (2014). México: Universidad Autónoma Metropolitana. Recuperado de http://pywoxserializer.sourceforge.net/.

World Wide Web Consortium - W3C (2016). Extensible Markup Language (XML). Recuperado de https://www.w3.org/XML/.

Zdun, U., Voelter, M., Kircher, M. (2004). Pattern-Based Design of an Asynchronous Invocation Framework for Web Services. *International Journal of Web Service Research*, 1(3). doi: 10.4018/jwsr.2004070103.