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

Introducción

La comunicación asíncrona en aplicaciones cliente-servidor es utilizada para procesos que necesitan ser completados por un servidor y que toman una cantidad de tiempo considerable. Si una aplicación cliente ejecuta un proceso largo sin utilizar comunicación asíncrona, la aplicación cliente se bloqueará mientras el servidor se encuentra ejecutando el proceso; y se desbloqueará cuando el proceso termine, es decir, cuando el servidor mande la respuesta de regreso a la aplicación cliente. Utilizando la comunicación asíncrona, un programa cliente puede continuar trabajando sin bloquearse, mientras el proceso está siendo ejecutado por el servidor.

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

Ejemplos de aplicaciones que requieren el uso de la comunicación asíncrona son los siguientes: un programa que solicita ejecutar un algoritmo complejo, el cual puede tomar varios minutos para ser completado; y la ejecución de una aplicación de reconocimiento de patrones, la cual puede tomar un largo tiempo para realizar los procesos de entrenamiento y reconocimiento.

En aplicaciones que operan en Internet, el protocolo de transferencia de hipertexto (*Hypertext Transfer Protocol*, HTTP, por sus siglas en inglés) (Fielding *et al.*, 2007) es el que más se utiliza para que se comuniquen; sin embargo, éste no proporciona una forma para abrir conexiones a clientes y notificarles cuando un proceso ha terminado. Como parte de la investigación previa que se ha realizado, se desarrolló un *framework* para programación de objetos distribuidos en los lenguajes de programación Java y C#, llamado Objetos Web en XML (*Web Objects in XML*, WOX, por sus siglas en inglés), el cual ha sido utilizado para implementar aplicaciones basadas en objetos distribuidos que operan en Internet, tales como las que se presentan en Jaimez-González y Lucas (2007). Este *framework* utiliza HTTP como protocolo de transporte, y los objetos los representa en el lenguaje de marcado extensible (*eXtensible Markup Language*, XML, por sus siglas en inglés) (W3C, 2016), y proporciona comunicación síncrona y asíncrona entre clientes y servidores.

Dada la importancia de la comunicación asíncrona en las aplicaciones distribuidas, en este artículo se presenta *AsyncMonitor*, una herramienta que permite monitorear las invocaciones de métodos asíncronos que se ejecutan en el *framework* WOX, al mismo tiempo que apoya la enseñanza de los conceptos de la comunicación asíncrona en cursos de sistemas distribuidos, debido a que proporciona una interfaz gráfica para visualizar los procesos que se encuentran ejecutándose en sus diferentes estados.

El resto del artículo está organizado de la siguiente manera. La siguiente sección presenta una introducción al *framework* WOX. Posteriormente se explican los dos mecanismos para realizar comunicación asíncrona: sondeo HTTP y transmisión HTTP. En dos secciones posteriores se describen las implementaciones de las invocaciones de métodos asíncronos utilizando sondeo y transmisión. Una siguiente sección está dedicada a presentar *AsyncMonitor*, la herramienta web de apoyo a la enseñanza para monitorear las invocaciones de métodos asíncronos. La penúltima sección presenta las pruebas de funcionalidad que se realizaron y los resultados obtenidos. Finalmente, se proporcionan conclusiones y trabajo futuro.

Objetos Web en XML

Esta sección proporciona una breve introducción al *framework* WOX, el cual combina características de los sistemas distribuidos basados en objetos y los sistemas distribuidos basados en web. Algunas de las características de WOX se presentan en los siguientes párrafos.

WOX utiliza localizadores uniformes de recursos (*Uniform Resource Locators*, URL, por sus siglas en inglés) para identificar objetos remotos de manera única, siguiendo los principios de la transferencia de estado representacional (*Representational State Transfer*, REST, por sus siglas en inglés) (Fielding, 2000). Esta característica es muy importante, ya que todos los objetos son identificados de manera única utilizando su URL y se pueden acceder desde cualquier lugar en la web, ya sea mediante un navegador o a través de una aplicación.

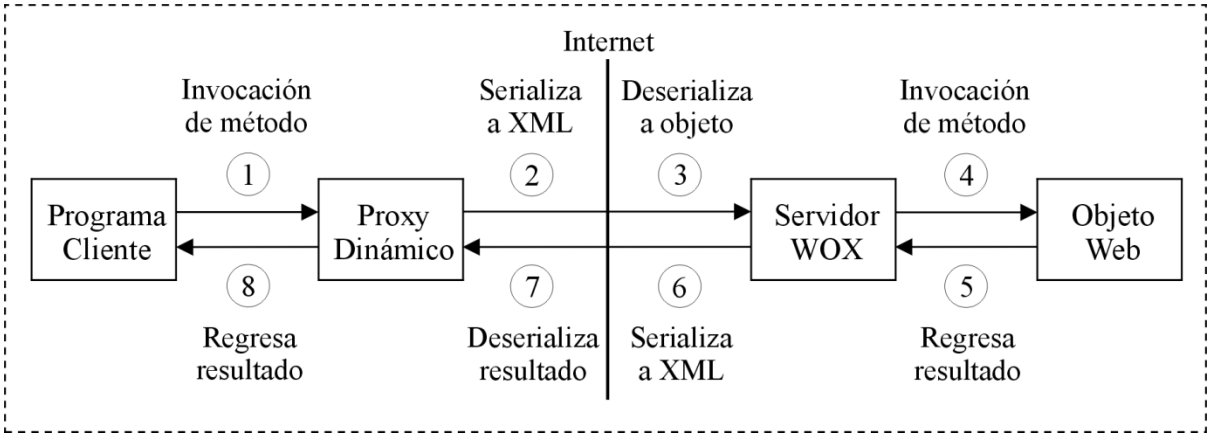
Este *framework* utiliza un serializador eficiente y de fácil uso llamado el serializador WOX (Jaimez-González y Lucas, 2011a; Jaimez-González, Lucas y López-Ornelas, 2011; WOX, 2009), el cual es la base del *framework* para serializar objetos, solicitudes y respuestas intercambiadas entre clientes y servidores. El serializador es una biblioteca independiente basada en XML, la cual es capaz de serializar objetos Java y C# a XML, así como tomar la representación XML para generar de nuevo los objetos Java y C#. Una de sus principales características es la generación de XML estándar para objetos, el cual es independiente del lenguaje de programación y permite interoperabilidad entre diferentes lenguajes de programación orientados a objetos. Aplicaciones desarrolladas en los lenguajes de programación Java y C# pueden interoperar mediante este *framework*. Cabe señalar que también se han desarrollado dos serializadores adicionales: uno en el lenguaje de programación Python, llamado PyWOX (PyWOX, 2014), y otro en el lenguaje de programación PHP, llamado PHPWOX (Hernández-Piña y Jaimez-González, 2016; PHPWOX, 2014).

WOX tiene operaciones estándar y especiales que se pueden aplicar sobre objetos remotos y locales, entre las cuales se incluyen las siguientes: la solicitud de referencias remotas, invocaciones de métodos estáticos (llamadas de servicios web), invocaciones de métodos de instancia, destrucción de objetos, solicitud de copias de objetos, duplicación de objetos, actualización de objetos, registro de objetos e invocaciones de métodos asíncronos. Algunas de estas operaciones se describen en Jaimez-González y Lucas (2007).

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

Un ejemplo del mecanismo utilizado por WOX en una invocación a un método de un objeto remoto se ilustra en la figura 1, y los pasos que se llevan a cabo se detallan a continuación.

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



Fuente: elaboración propia.

1. El programa cliente WOX invoca un método sobre una referencia remota (la forma en la cual el cliente invoca un método sobre una referencia remota es exactamente la misma forma en la que lo hace sobre un objeto local).
2. El *proxy* dinámico WOX toma la solicitud, la serializa a XML, y la manda a través de la red al servidor WOX.
3. El servidor WOX toma la solicitud y la deserializa a un objeto WOX.
4. El servidor WOX carga el objeto en memoria y ejecuta el método solicitado.
5. El resultado de la invocación del método es regresado al servidor WOX.
6. El servidor WOX serializa el resultado a XML y regresa al cliente el resultado real o una referencia a ese resultado. El resultado se almacena en el servidor en el caso de que se regrese una referencia.
7. El *proxy* dinámico WOX recibe el resultado y lo deserializa al objeto adecuado, ya sea objeto real o referencia remota.
8. El *proxy* dinámico WOX regresa el resultado al programa cliente.

Desde el punto de vista del programa cliente, se realiza la invocación del método y se obtiene el resultado de regreso de una forma transparente. Las bibliotecas cliente WOX llevan a cabo el proceso de serializar la solicitud y mandarla al servidor WOX, así como recibir el resultado de la invocación del método y deserializarlo.

WOX también proporciona una interfaz web, la cual permite la inspección de objetos a través de un navegador web, la ejecución de métodos sobre objetos, y la visualización de objetos con tres diferentes modos de operación: xml, html e imagen (Jaimez-González, 2014). También tiene la característica de almacenar y navegar objetos remotos, lo cual permite a un programa cliente recuperar objetos hijo de un objeto raíz dado, a través de su representación XML (Hernández-Salinas y Jaimez-González, 2016).

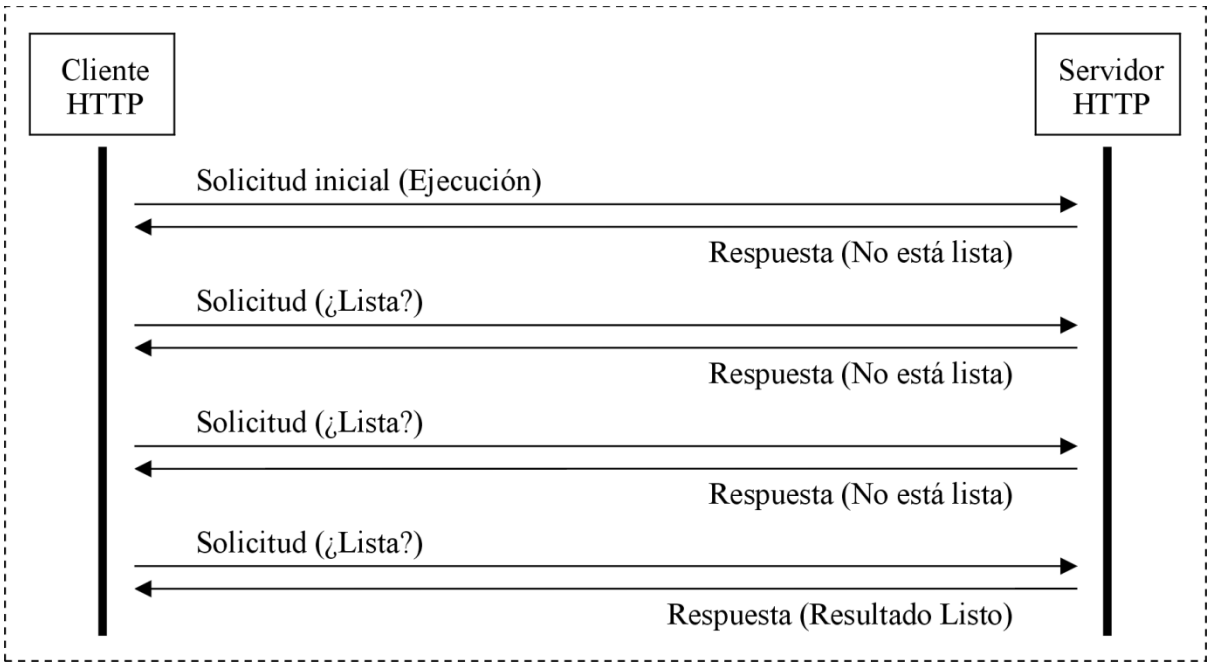
Las siguientes secciones se concentran en la herramienta web para apoyar la enseñanza de la comunicación asíncrona, con la cual es posible monitorear las invocaciones de métodos asíncronos en WOX; así como también se describe la implementación que se llevó a cabo de la comunicación asíncrona utilizando sondeo y transmisión HTTP, tomando como base la propuesta de Jaimez-González y Lucas (2011b).

Técnicas de sondeo HTTP y transmisión HTTP

La figura 2 muestra la técnica de sondeo HTTP, en la cual el cliente periódicamente hace solicitudes al servidor para obtener información acerca de su petición original. Cada llamada está separada por un periodo de tiempo. En esta técnica la primera llamada del cliente es realmente para hacer la solicitud original de ejecución del algoritmo o programa, y las llamadas subsecuentes son para sondear si la solicitud original ha terminado su ejecución. Para cada llamada el servidor responderá inmediatamente ya sea con la respuesta del proceso completado o con la indicación de que el proceso aún no ha terminado. Esta técnica simula un canal de comunicación en segundo plano entre el cliente y el servidor, y es muy parecido a la situación en la que el servidor manda datos directamente al cliente, como si el servidor realmente abriera una conexión hacia el cliente para notificarle cuando el proceso ha terminado.

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

Figura 2. Comunicación asíncrona con técnica de sondeo HTTP.



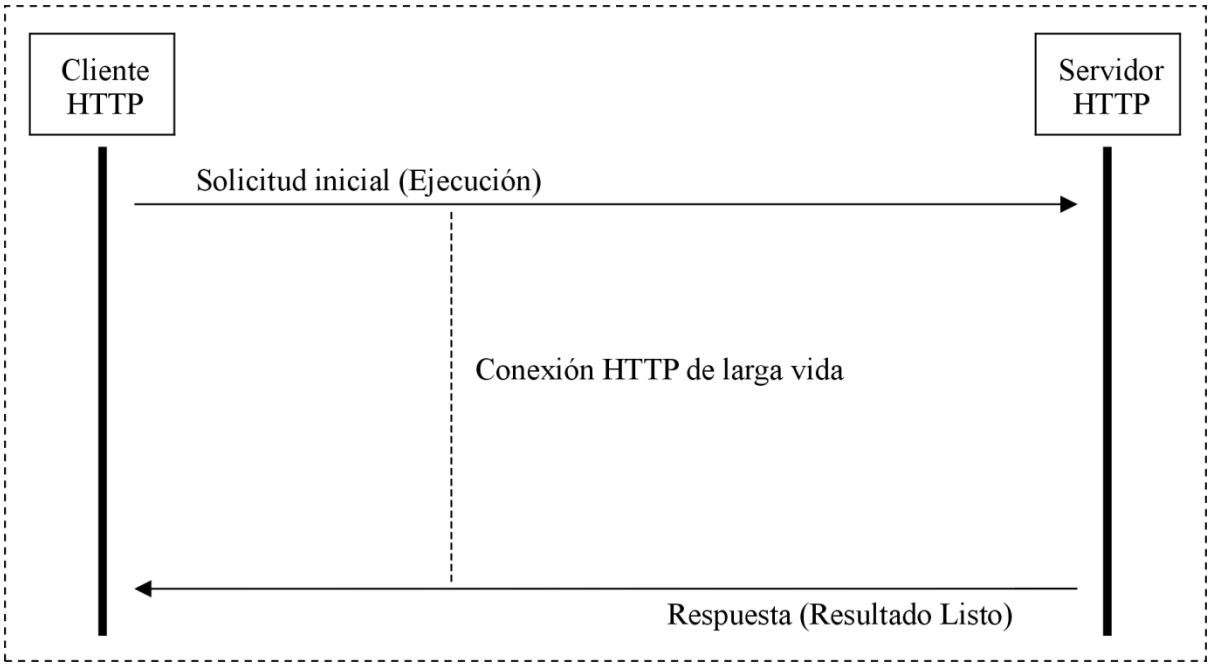
Fuente: elaboración propia.

Este enfoque tiene un costo significativo debido a las llamadas periódicas al servidor, ya que cada llamada, sin importar lo pequeña que pueda ser, demanda recursos tanto en el cliente como en el servidor; cada solicitud tiene también un costo de ancho de banda asociado. Un diseño clave de este enfoque sería incrementar el promedio de tiempo del periodo en que se hacen las llamadas o sondeos, mientras se mantiene una buena experiencia para el cliente; esto depende del tipo de aplicación, pero suponiendo que se sabe que el algoritmo o proceso puede tomar unos pocos minutos para terminar, cada llamada del cliente podría realizarse cada 30 segundos o un minuto, por ejemplo; esto también dependerá de los requerimientos de la aplicación y el número de usuarios que accedan a ella.

La técnica de transmisión HTTP se muestra en la figura 3, en la cual se observa que se utiliza una conexión HTTP de larga vida. El cliente manda la solicitud al servidor, el cual mantiene la conexión abierta; cuando el proceso ha terminado su ejecución o hay actualizaciones que deban ser notificadas a la aplicación cliente, el servidor envía la respuesta o las actualizaciones al canal de salida. Este enfoque tiene algunas desventajas: las conexiones de larga vida inevitablemente fallarán, por lo que debe existir un plan de recuperación en caso de que la

conexión se rompa; otra desventaja es que los servidores no pueden mantener muchas conexiones simultáneas abiertas, las cuales son necesarias si se utiliza esta solución. La conexión debe mantenerse abierta por un periodo razonable de tiempo, y esto dependerá de los recursos involucrados, tales como el servidor, la red y el número de clientes conectados en un determinado momento, entre otros aspectos.

Figura 3. Comunicación asíncrona con técnica de transmisión HTTP.



Fuente: elaboración propia.

Voelter, Kiercher y Zdun (2003) proponen un diseño de invocaciones asíncronas en *frameworks* de objetos distribuidos, el cual también fue utilizado para invocaciones asíncronas en *framewors* de servicios web (Zdun, Voelter y Kircher, 2004). Bai y Tao (2010) proponen un intermediario de mensajes utilizando invocación de métodos asíncronos con servicios web. En el caso del *framework* WOX, se implementaron la transmisión HTTP y el sondeo HTTP. Dado que WOX permite las invocaciones de métodos sobre objetos web remotos a través de HTTP, la transmisión HTTP fue implementada mediante invocaciones de métodos asíncronos para permitir que una aplicación cliente ejecute remotamente un método sobre un objeto que reside en un

servidor WOX. Cuando el método termina su ejecución, la aplicación cliente será contactada y una acción específica, definida por el cliente, podría ser ejecutada.

Para el caso del sondeo HTTP la implementación en WOX fue también a través de invocaciones de métodos asíncronos o procesos que se ejecutan en un servidor WOX. En este enfoque el proceso también se ejecuta en el servidor y cuando este termina, el resultado es almacenado como un objeto web en el servidor, y expuesto como cualquier otro objeto a través de su propia URL. De esta forma, el cliente puede recuperar el resultado de la invocación del método asíncrono como lo hace para cualquier otro objeto. Dado que un objeto en WOX es representado en XML, y puede ser accesado utilizando cualquier navegador web, este enfoque es muy conveniente, ya que el resultado puede ser recuperado desde cualquier lugar en Internet. Para mayor información acerca de la representación XML para objetos Java y C# en WOX puede consultarse Jaimez-González, Lucas y López-Ornelas (2011), o puede visitarse el sitio web del proyecto *Web Objects in XML* (WOX, 2009), de donde puede ser descargado.

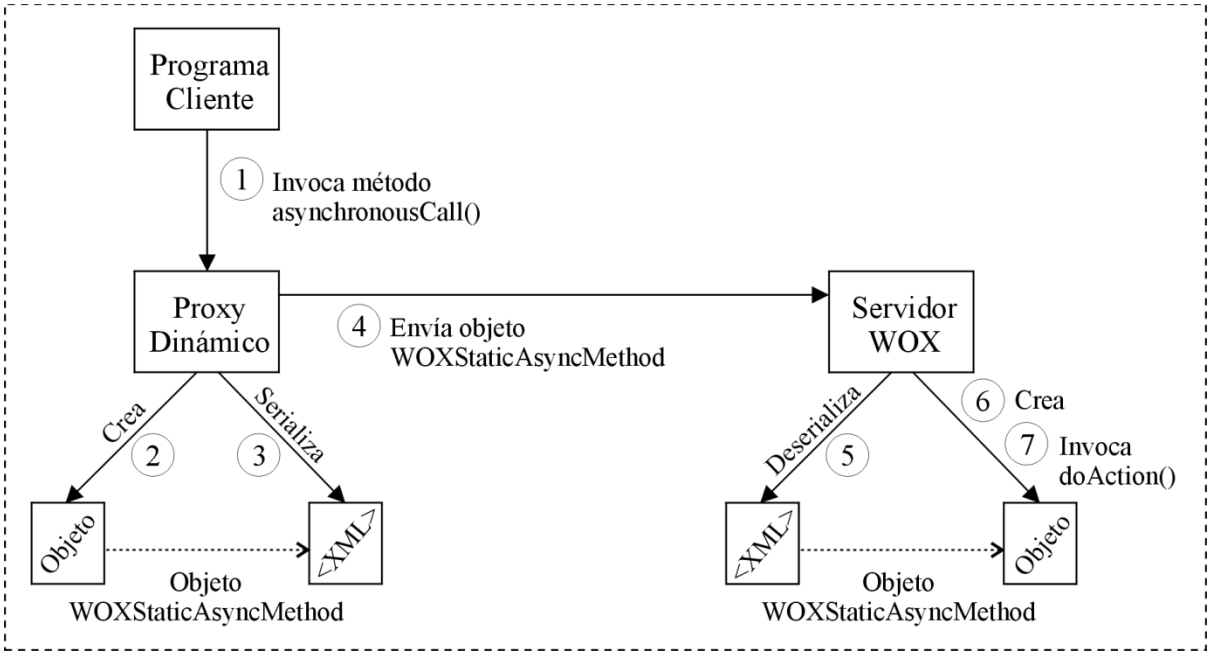
Comunicación asíncrona utilizando sondeo

La comunicación asíncrona utilizando la técnica de sondeo es llevada a cabo a través de invocaciones de métodos asíncronos en el *framework* WOX. Una aplicación cliente invoca métodos asíncronos sobre objetos remotos, los cuales residen en un servidor. Esta sección explica el proceso de invocar tales métodos desde una aplicación cliente, cómo son ejecutados en el servidor, y cómo son implementados en WOX. Cabe señalar que el lenguaje de programación Java es utilizado para el código mostrado en esta sección.

Las invocaciones de métodos asíncronos son utilizadas cuando un método o proceso toma una cantidad de tiempo considerable para terminar; evitan que el cliente se bloquee, esperando la ejecución del método en el servidor. En las figuras 4 y 5 se muestran los pasos llevados a cabo para una invocación de método asíncrono en WOX utilizando la técnica de sondeo, y a continuación se proporciona una explicación de cada paso.

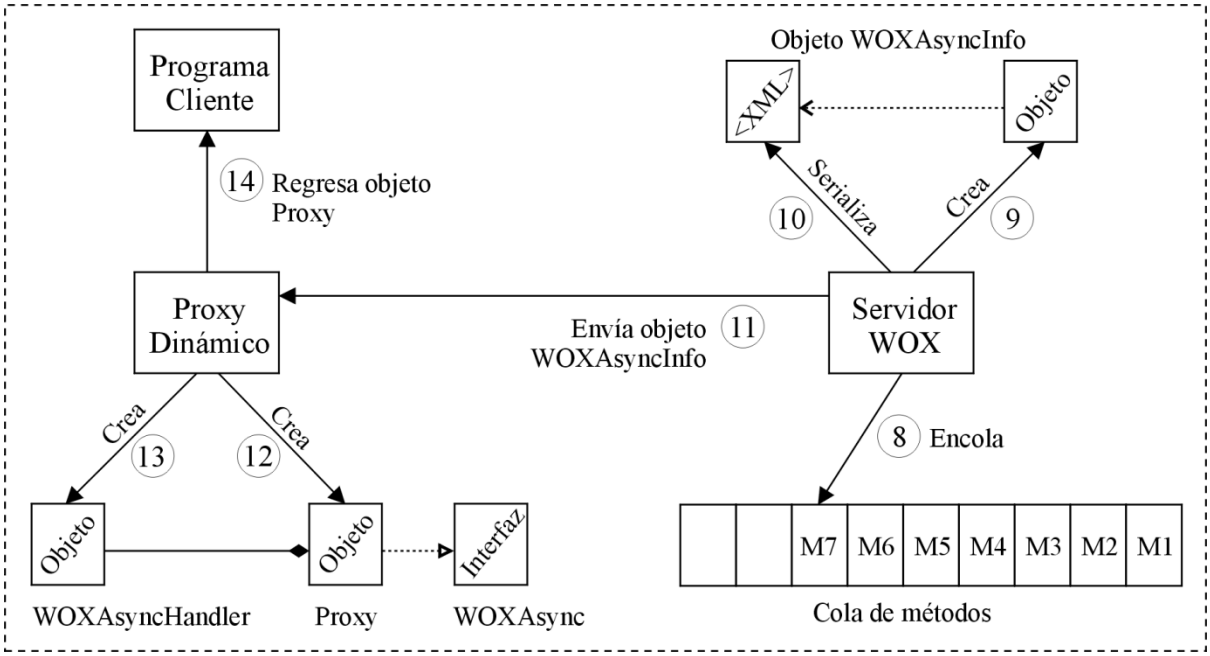
Revista Iberoamericana de las Ciencias Computacionales e Informática
ISSN: 2007 - 9915

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. En el paso 1 de la figura 4, el cliente invoca un método estático sobre una clase que ya reside en el servidor; también podría ser un método de instancia de un objeto existente. Esto se realiza mediante la clase *WOXProxy* invocando el método *asynchronousCall()*, el cual tiene dos variantes, dependiendo del tipo de método, estático o de instancia. La variante estática, similar a una invocación de método estático, se muestra abajo, en la cual se observa que toma como parámetros: la URL del servidor, la clase donde se encuentra el método, el método que será invocado, junto con cualquier parámetro que sea necesario. Cabe señalar que el tipo de retorno del método *asynchronousCall()* es un objeto *WOXAsync*.

   WOXAsync woxAsync = (WOXAsync) WOXProxy.asynchronousCall (serverURL, className, methodName, params);
2. El paso 2 de la figura 4 representa que las bibliotecas cliente de WOX transforman la llamada del método *asynchronousCall()* en un objeto *WOXStaticAsyncMethod*.
3. Las bibliotecas cliente de WOX serializan el objeto *WOXStaticAsyncMethod* a XML, como se muestra en el paso 3 de la figura 4.
4. El paso 4 de la figura 4 corresponde al envío del objeto *WOXStaticAsyncMethod* serializado al servidor WOX, de la misma forma como se envía cualquier otra solicitud de un cliente.
5. El servidor recibe el objeto *WOXStaticAsyncMethod* y lo deserializa para convertirlo en un objeto, como se muestra en el paso 5 de la figura 4.
6. En el paso 6 de la figura 4 el servidor se encarga de crear el objeto a partir de la deserialización que se llevó a cabo en el paso anterior.
7. El servidor ejecuta el método *doAction* del objeto, como se observa en el paso 7 de la figura 4, de la misma forma como se ejecuta cualquier otra solicitud.
8. La ejecución del método *doAction* colocará la solicitud del método asíncrono en una cola para su ejecución posterior, en donde un hilo se encarga de dicha cola de métodos, como se observa en el paso 8 de la figura 5.
9. En el paso 9 de la figura 5 el servidor creará un objeto *WOXAsyncInfo*, el cual contiene suficiente información para rastrear la ejecución del método que se encuentra en la cola, tal como el identificador del proceso que le fue asignado.

10. El objeto *WOXAsyncInfo* es serializado a XML por el servidor, como se observa en el paso 10 de la figura 5.
11. El paso 11 de la figura 5 muestra el envío del objeto *WOXAsyncInfo* serializado al cliente, el cual le servirá para monitorear la ejecución del método que ya se encuentra en la cola.
12. El programa cliente, a través de las bibliotecas cliente de WOX, recibe el objeto *WOXAsyncInfo* y crea un objeto *Proxy*, como se muestra en el paso 12 de la figura 5, el cual implementa la interfaz *WOXAsync*. Esta interfaz contiene métodos para monitorear la ejecución de los métodos y procesos asíncronos que se encuentran en la cola de ejecución.
13. En el paso 13 de la figura 5 el servidor crea un objeto *WOXAsyncHandler* para manejar todas las invocaciones de métodos sobre el objeto *Proxy*. Cada llamada sobre la instancia del *proxy* será manejada por este objeto, el cual contactará al servidor para recabar información acerca del método. El objeto *Proxy* está compuesto de un objeto *WOXAsyncHandler*, como se muestra en la figura 5.
14. El objeto *Proxy* es regresado al cliente, como se observa en el paso 14 de la figura 5, con el cual podrá monitorear la ejecución del método asíncrono que se encuentra en la cola de métodos del servidor.

Una vez que el método *asynchronousCall* ha regresado, el cliente será capaz de monitorear el método asíncrono mediante los métodos proporcionados por la interfaz *WOXAsync*. El cliente puede invocar los siguientes métodos sobre el proxy *woxAsync*:

getProcessId(). Este método regresa el identificador del proceso *processId*, dado por el servidor al método. Esta información puede ser obtenida del objeto *WOXAsyncInfo*, el cual es enviado inmediatamente al cliente.

getStatus(). Este método regresa un entero que representa el estatus de la invocación del método: pendiente (*Pending*), el cual indica que el método se encuentra aún en la cola esperando ser ejecutado; activo (*Active*), el cual indica que el método está siendo ejecutado en este momento; o completado (*Completed*), el cual indica que el método ha terminado y el resultado está listo para que sea recuperado.

getResultAsReference(). Este método puede ser ejecutado solamente cuando el método *getStatus()* ha regresado *Completed*, lo cual significa que el resultado de la ejecución del método está listo para que sea recuperado. Este método obtendrá una referencia al resultado de la ejecución del método.

getResultAsCopy(). Este método es parecido al método *getResultAsReference()*, pero el método *getResultAsCopy()* no recupera una referencia al resultado de la ejecución del método, sino una copia del resultado.

Para los métodos que se mencionaron, excepto el método *getProcessId()*, el cliente enviará la solicitud al servidor como un objeto *WOXMonitor*. El servidor recibirá esta solicitud y la procesará de la misma forma como cualquier otra solicitud de un cliente; es decir, el servidor ejecutará el método *doAction()* del objeto *WOXMonitor*, el cual utilizará el *processId* para identificar el proceso y regresar una respuesta adecuada al cliente.

Cada solicitud que un cliente realiza para invocar un método asíncrono es recibida por el servidor y encolada; hay un hilo que ejecuta los métodos en la cola secuencialmente. Cuando un método ha terminado su ejecución, su resultado es almacenado como cualquier otro objeto en el *framework* WOX, y una referencia a este es almacenada en la cola, lo cual permite al servidor proporcionar una referencia al objeto cuando el cliente la solicite. La siguiente información es almacenada para cada solicitud de método asíncrono: *processId*; método que será ejecutado, objeto *WOXMethod*, método estático o de instancia; estatus de la ejecución del método (pendiente, activo o completado); y una referencia al resultado de la ejecución del método, mediante un objeto *WOXReference*.

Como trabajo futuro en esta implementación, es necesario incluir un atributo *processURL* a la información almacenada en cada solicitud de método asíncrono, el cual representaría un URL único donde cada proceso estaría siendo monitoreado. También podría incluirse el porcentaje completado del proceso para saber el avance que tiene, ya que en este momento solamente se reporta el estatus del proceso: pendiente, activo o completado.

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

Comunicación asíncrona utilizando transmisión

Esta sección presenta la otra implementación de comunicación asíncrona, la cual utiliza la técnica de transmisión, donde las bibliotecas cliente WOX son las encargadas de verificar si el proceso ha terminado, no es el programa cliente quien lo hace, de tal forma que cuando el proceso termina su ejecución en el servidor, el cliente es notificado y una acción puede ser ejecutada. También se utiliza el lenguaje de programación Java para el código mostrado en esta sección. La acción que se ejecutará es especificada por un método en el programa cliente, y las bibliotecas cliente proporcionan el método *asynchronousCall* para este propósito, como se muestra a continuación.

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

Los parámetros del método *asynchronousCall* son los siguientes: *serverURL* es el URL donde se encuentra el servidor; *className* es la clase en donde está el método que será invocado; *methodName* es el nombre del método que será invocado en el servidor; *params* representa los parámetros del método que será invocado; *objForCallbak* es una referencia local del objeto que tiene el método *callbackMethod* que será invocado cuando el método asíncrono en el servidor termine su ejecución.

En la implementación de la técnica de transmisión HTTP que se realizó, la aplicación cliente invoca el método *asynchronousCall* mostrado previamente, y puede continuar trabajando sin bloquearse. En este enfoque, la implementación del método *asynchronousCall* se ejecuta en un proceso en segundo plano, el cual monitorea que termine el proceso que se encuentra ejecutándose en el servidor. La aplicación cliente no se percata que hay un proceso ejecutándose en segundo plano. El proceso que se encarga de monitorear el servidor es muy semejante al que se describió previamente en la técnica de sondeo, pero con la técnica de transmisión las bibliotecas cliente de WOX implementan esta funcionalidad en segundo plano.

Cuando el proceso ha terminado su ejecución en el servidor, el proceso que se encuentra en segundo plano monitoreando su terminación, solicita el resultado y lo recupera; con la técnica de sondeo, esto era llevado a cabo explícitamente por el programa cliente. El proceso que se ejecuta en segundo plano invoca el método sobre el objeto local que fue especificado por el programa cliente, es decir, el método *callbackMethod* del objeto *objectForCallback*; lo cual es otra diferencia con respecto a la implementación de la técnica de sondeo, donde el programa cliente

solamente recupera el resultado. En la implementación de la técnica de transmisión, las bibliotecas cliente recuperan el resultado, el cual está disponible al programa cliente, pero también se ejecuta un método sobre un objeto local.

En resumen, en la implementación de la técnica de sondeo, el programa cliente es responsable de monitorear el proceso asíncrono hasta que termine, y cuando el proceso ha terminado su ejecución, el programa cliente puede solicitar una referencia o una copia del resultado de la ejecución del método, lo cual es llevado a cabo por la interfaz *WOXAsync*. En la implementación de la técnica de transmisión, un proceso en segundo plano, el cual es parte de las bibliotecas cliente de WOX, es el responsable de monitorear el proceso asíncrono hasta que termine, de tal forma que cuando termina su ejecución en el servidor, el programa cliente es notificado, y una acción definida por el programa cliente puede ser ejecutada.

AsyncMonitor

El *framework* WOX incluye una herramienta web, llamada *AsyncMonitor*, para métodos y procesos asíncronos, la cual permite a los clientes monitorear la cola de procesos que están siendo ejecutados en el servidor, y a su vez apoya en la enseñanza de la comunicación asíncrona en cursos de sistemas distribuidos, ya que muestra gráficamente cómo se ejecutan los procesos asíncronos en una cola y permite visualizar el estado de cada uno de ellos.

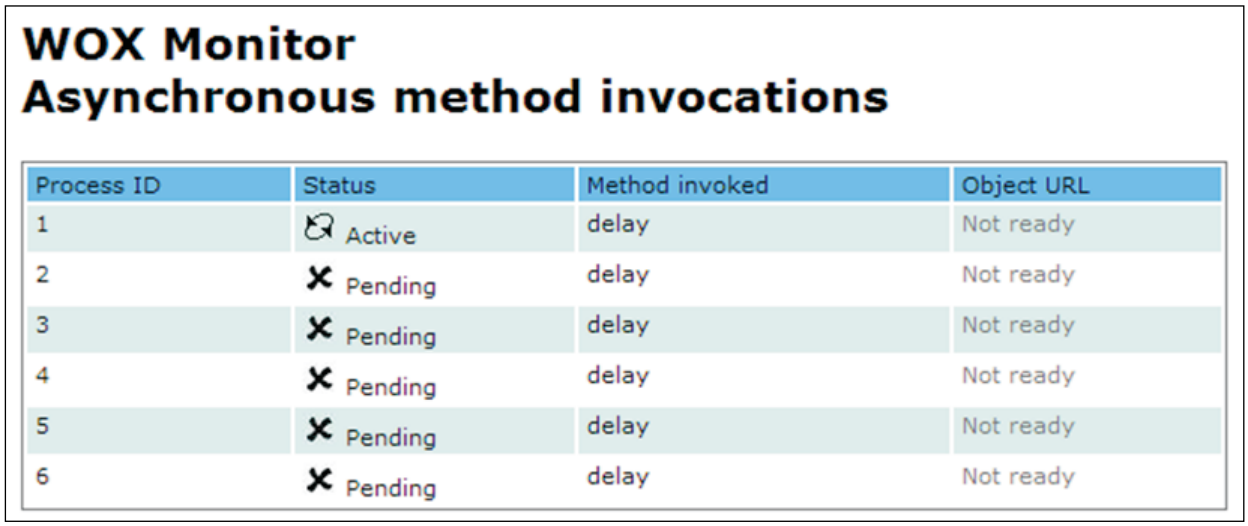
Se puede acceder a *AsyncMonitor* a través de un navegador web, en el cual es posible visualizar información acerca de los métodos y procesos que están siendo ejecutados actualmente, ya que proporciona lo siguiente: el identificador del proceso; el estatus del proceso (pendiente, activo o completado); el nombre del proceso; y el URL del resultado del proceso, el cual es en realidad un objeto, solamente cuando el método ha terminado su ejecución. Esta herramienta puede ser utilizada para rastrear las invocaciones de métodos asíncronos recibidas en un servidor WOX en particular y puede ser consultada en la URL mostrada abajo, donde *localhost* debe ser reemplazado por la dirección IP o por el nombre de la computadora donde se instaló el servidor WOX:

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

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

La figura 6 muestra una captura de pantalla de *AsyncMonitor*, después de haber recibido seis solicitudes de clientes que desean ejecutar métodos asíncronos. El estatus del primer proceso es activo (*Active*), es decir, este proceso está siendo ejecutado por el servidor; el resto de los procesos tienen el estatus de pendiente (*Pending*), lo cual significa que están pendientes de ejecución como se observa por las URL de los objetos que se encuentran como *Not ready*, ya que todos estos métodos no han terminado su ejecución.

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

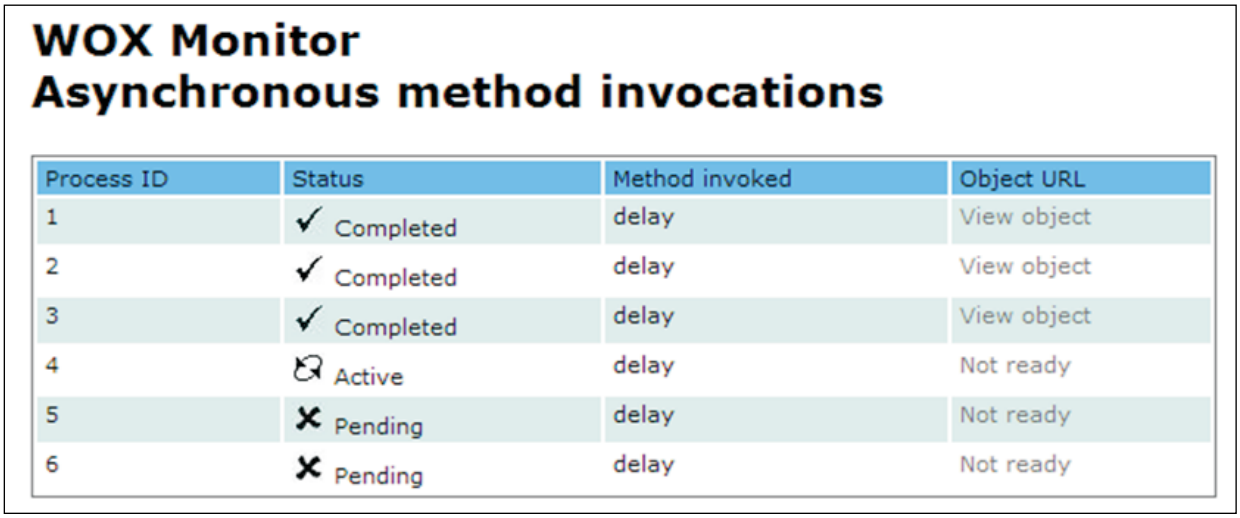


Fuente: elaboración propia.

La figura 7 muestra una captura de pantalla de *AsyncMonitor* un periodo de tiempo después de la que se muestra en la figura 6. Los primeros tres procesos o invocaciones de métodos han terminado su ejecución. Los resultados de las ejecuciones de esos métodos están listos y pueden accederse mediante el hipervínculo *View object*, el cual es un URL que lleva al resultado de la invocación del método. Los resultados pueden también recuperarse mediante un programa cliente con los métodos *getResultAsCopy()* o *getResultAsReference()*, los cuales se presentaron en una sección previa. El cuarto método que se muestra en la figura 7 está aún activo, y el resto de los métodos están pendientes de ejecución.

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 es una herramienta web muy útil para visualizar todas las invocaciones de métodos asíncronos que están siendo ejecutados en un servidor WOX. Cabe señalar que un programa cliente podría también monitorear las invocaciones de métodos asíncronos mediante programación, utilizando los métodos presentados previamente.

La cola de métodos en *AsyncMonitor* está implementada como un mapa en memoria, ya que de esta forma es de fácil y rápido acceso y los resultados de las invocaciones a métodos están almacenados como objetos. Existe la posibilidad de almacenar el mapa en una base de datos o en algún otro medio de almacenamiento cada vez que el mapa es modificado, lo cual sucedería en alguno de los siguientes escenarios:

1. La llegada de una nueva solicitud de ejecución de un método asíncrono, ya que se requiere un nuevo registro en el mapa.
2. El comienzo de la ejecución de un método, ya que el estatus del proceso cambia de pendiente a activo.
3. La terminación de un método, ya que el resultado es almacenado como un objeto y una referencia al resultado de la ejecución es añadido al mapa.

El mapa se mantendría actualizado; si el servidor se apaga en algún momento, el mapa puede ser recuperado cuando el servidor inicie nuevamente, de tal forma que el servidor necesitaría reiniciar con el proceso que se haya quedado marcado como activo.

La ejecución de los métodos en la cola se implementó con un hilo que se encarga de los procesos, el cual los ejecuta secuencialmente en el orden en el cual se encuentran en la cola, es decir, el orden en el que han llegado. El hilo se va a dormir cuando no hay más procesos que deban ser ejecutados en la cola, y despierta cuando un nuevo proceso llega para ser ejecutado.

Existen otros enfoques para implementar la ejecución de métodos en la cola; por ejemplo, se podría contemplar un número fijo de hilos, los cuales se encarguen de los procesos en la cola, donde cada hilo podría ser creado conforme se vaya necesitando. Otro enfoque sería crear un hilo para cada solicitud de invocación de método asíncrono que se reciba, de tal forma que cada solicitud sea atendida tan pronto como llegue; sin embargo, la desventaja es que tendrían que crearse tantos hilos como solicitudes concurrentes hubiera, además de los recursos de procesamiento que deberían existir en el servidor que atendiera estas peticiones.

Una situación importante que debe considerarse en la ejecución de métodos en la cola es su prioridad, la cual podría ser asignada por la aplicación cliente al momento de la solicitud de ejecución. La definición de políticas para este tipo de problemas es parte del trabajo a futuro que se realizará para esta herramienta de apoyo.

Pruebas y resultados

En esta sección se presentan las pruebas que se realizaron y los resultados obtenidos. Se llevaron a cabo pruebas de funcionalidad de *AsyncMonitor*, con el apoyo de alumnos y estas fueron con diferentes escenarios que se explican a continuación.

El objetivo de las pruebas fue verificar la funcionalidad de *AsyncMonitor* para diferentes tareas: recibir solicitudes de invocaciones de métodos asíncronos, la correcta ejecución de los métodos con sus estatus correspondientes, la visualización de los objetos resultantes, así como la funcionalidad de los hilos de ejecución utilizando los dos mecanismos de comunicación asíncrona (sondeo HTTP y transmisión HTTP).

Para las pruebas se utilizaron seis computadoras. Se instaló el servidor WOX en una computadora con las siguientes características: sistema operativo Windows 10 Professional de 64

bits, procesador Intel Core i7-7700K a 4.50 GHz, memoria RAM de 16 GB. Las cinco computadoras que se utilizaron como clientes tenían las siguientes características: sistema operativo Windows 7 Professional de 64 bits, procesador Intel Core i7-2760 a 2.40 GHz, memoria RAM de 8 GB.

Se realizaron un total de 10 escenarios de pruebas para cada mecanismo de comunicación asíncrona, es decir, un total de 20 escenarios: 10 utilizando sondeo HTTP y 10 utilizando transmisión HTTP. En cada escenario se mantuvo disponible el servidor WOX y las cinco computadoras cliente empezaron a enviar un número aleatorio de solicitudes de invocaciones de métodos asíncronos al servidor WOX. En cada computadora cliente se tenía ejecutándose un programa en el lenguaje Java que realizaba invocaciones de métodos asíncronos sobre objetos que residían en el servidor WOX, de tal manera que para enviar las solicitudes de invocación de métodos utilizaba las bibliotecas cliente de WOX, las cuales se encargaban de contactar directamente al servidor WOX a través de un *Proxy*. En cada computadora cliente se accedió a *AsyncMonitor* a través de un navegador web para verificar la información de sus solicitudes.

En la tabla 1 se muestran los resultados del primer escenario utilizando el mecanismo de sondeo HTTP, los cuales se obtuvieron mediante la interacción de las cinco computadoras cliente con el servidor WOX. Cada renglón de la tabla representa los resultados de las invocaciones de una computadora cliente en particular (Cliente1, Cliente2, Cliente3, Cliente4 y Cliente5). Con respecto a las columnas, la primera de ellas indica el número de solicitudes de invocación de métodos que envió esa computadora cliente al servidor; la segunda columna contiene el número de esas solicitudes que fueron encoladas en la cola de métodos que se encuentra en el servidor; la tercera columna representa el número de solicitudes que terminaron exitosamente su ejecución; la cuarta columna indica cuántas de las solicitudes enviadas cambiaron correctamente su estatus (pendiente, activo y completado); y, finalmente, la quinta columna muestra para cuántas de las solicitudes fue posible visualizar el resultado de su ejecución como un objeto.

En el primer renglón de la tabla 1, puede observarse que el Cliente1 envió cinco solicitudes de invocación de métodos asíncronos; cinco solicitudes fueron encoladas en el servidor; solamente tres métodos terminaron su ejecución, debido a que dos de ellos resultaron en excepciones de programa, por lo cual no terminaron su ejecución; tres solicitudes cambiaron correctamente sus estatus de pendiente a activo y de activo a completado (las dos solicitudes que

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

resultaron en excepción se quedaron erróneamente con el estatus de activo); los resultados de la ejecución de los tres métodos que terminaron correctamente fueron visualizados en el navegador web en su representación XML. De forma similar pueden leerse los resultados del Cliente 2, 3, 4 y 5. Para cada uno de los 10 escenarios se generó una tabla como la que se muestra en la tabla 1, en donde se especificaron de manera aleatoria el número de solicitudes que cada computadora cliente envió al servidor WOX, y se capturó la información.

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*.

En la tabla 2 se muestra el concentrado de los 10 escenarios que se realizaron utilizando la técnica de sondeo HTTP. Cabe señalar que el número de solicitudes que no fueron terminadas se debe a que los métodos no terminaron su ejecución por causas relacionadas con el código mismo del método, en todas ellas se dispararon excepciones propias del funcionamiento de los métodos, sin que esto tenga ninguna relación con el correcto funcionamiento de *AsyncMonitor*. Sin embargo, el cambio de estatus de aquellos métodos que no terminaron su ejecución exitosamente es un problema que debe corregirse, ya que hasta el momento cuando un método no termina su ejecución, su estatus se queda establecido en activo, dando la impresión de que el método sigue ejecutándose, cuando en realidad ya ha finalizado.

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

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*.

De manera similar a los resultados mostrados en la tabla 1, en la tabla 3 se muestran los resultados del primer escenario ahora utilizando el mecanismo de transmisión HTTP. En el primer renglón de la tabla, puede observarse que el Cliente1 envió siete solicitudes de invocación de métodos asíncronos; siete solicitudes fueron encoladas en el servidor; los siete métodos terminaron su ejecución; las siete solicitudes cambiaron correctamente sus estatus de pendiente a activo y de activo a completado; los resultados de la ejecución de los siete métodos que terminaron correctamente fueron visualizados en el navegador web en su representación XML. De forma similar pueden leerse los resultados del Cliente 2, 3, 4 y 5. Para cada escenario se generó una tabla como la que se muestra, en donde se especificaron de manera aleatoria el número de solicitudes que cada computadora cliente envió al servidor.

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*.

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

En la tabla 4 se muestra el concentrado de los 10 escenarios que se realizaron utilizando la técnica de transmisión HTTP. Cabe señalar que el número de solicitudes que no fueron terminadas (tercera columna) se debe a que los métodos que se ejecutaron no terminaron su ejecución, de manera similar a los resultados mostrados en la tabla 2; en todos los casos esto fue debido a que se dispararon excepciones propias del funcionamiento de los métodos particulares, sin que esto tenga ninguna relación con el correcto funcionamiento de *AsyncMonitor*. Con respecto al cambio de estatus de las solicitudes, ocurre lo mismo que en la tabla 2, donde para aquellos métodos que no terminaron su ejecución su estatus permaneció de manera errónea en activo; esto último es algo que deberá corregirse para la siguiente versión de *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*.

Los resultados obtenidos en estas pruebas de funcionalidad son alentadores, ya que permiten observar que hay un correcto funcionamiento de la ejecución de métodos asíncronos con *AsyncMonitor* del *framework* WOX. Del total de solicitudes enviadas para invocación de métodos asíncronos, 100% fueron encoladas en la cola de métodos del servidor. Si bien no todas las solicitudes terminaron su ejecución, esto no fue debido a algún problema con el servidor WOX, sino por el código mismo de los métodos que se enviaron a ejecutar. El estatus de los métodos que terminaron exitosamente cambió correctamente, mientras que los de aquellos métodos que no terminaron, permanecieron en activo, lo cual indica que debe trabajarse en este punto para que su estatus cambie de manera adecuada. La visualización de objetos también se

llevó a cabo de manera exitosa para aquellos métodos que terminaron su ejecución, mostrando los objetos en representación XML.

Conclusiones y trabajo futuro

En este artículo se presentó una introducción a un *framework* para programación de objetos distribuidos en Java y C#, llamado *Web Objects in XML* (WOX), el cual ha sido utilizado para implementar aplicaciones distribuidas basadas en objetos. WOX utiliza HTTP como protocolo de transporte, XML para la representación de objetos, y proporciona comunicación síncrona y asíncrona entre clientes y servidores.

Se describió la implementación de la comunicación asíncrona en WOX, en la cual se utilizaron dos mecanismos diferentes: sondeo HTTP y transmisión HTTP. Aunque no existe en HTTP forma de abrir conexiones a clientes para notificarles cuando un proceso ha terminado, se simuló la comunicación asíncrona en WOX, utilizando las técnicas de sondeo y transmisión; con lo cual fue posible tener una implementación robusta y muy útil, de tal forma que ahora no solamente se soportan las invocaciones de métodos síncronos sobre objetos remotos en WOX, sino también las invocaciones de métodos asíncronos.

También se presentó una herramienta web para monitorear procesos asíncronos en WOX, llamada *AsyncMonitor*, la cual también es considerada como herramienta de apoyo a la enseñanza de la comunicación asíncrona en cursos de sistemas distribuidos, ya que 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.

Se hicieron pruebas con el objetivo verificar la funcionalidad de *AsyncMonitor* para diferentes tareas: recibir solicitudes de invocaciones de métodos asíncronos, la correcta ejecución de los métodos con sus estatus correspondientes, la visualización de los objetos resultantes, así como la funcionalidad de los hilos de ejecución utilizando los dos mecanismos de comunicación asíncrona (sondeo HTTP y transmisión HTTP). Las pruebas contemplaron 20 escenarios: 10 utilizando el mecanismo de comunicación asíncrona de sondeo HTTP y 10 utilizando la técnica de transmisión HTTP. 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. En cada computadora cliente se accedió a *AsyncMonitor* para verificar la información de sus solicitudes.

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 la ejecución de los métodos que fueron encolados, 85% terminaron su ejecución; el resto de los métodos que no terminaron su ejecución (15%) fue debido a excepciones ocurridas por la implementación del método mismo que se mandó como solicitud, pero no fue debido al funcionamiento de *AsyncMonitor* ni del *famework* WOX. Con respecto a los estatus de los métodos encolados, cambiaron adecuadamente los de aquellos métodos que terminaron su ejecución exitosamente, es decir, cambiaron de *pendiente* a *activo* y de *activo* a *completado*. Para el caso de los estatus de los métodos que no terminaron su ejecución, permanecieron en *activo*, lo cual indica que hay un problema que debe atenderse para la siguiente versión de *AsyncMonitor*. La visualización de la representación XML de los objetos fue satisfactoria, para aquellos métodos que terminaron su ejecución exitosamente, ya que para los demás no pudo haber visualización dado que no hay un resultado que mostrar, debido a que el método no terminó su ejecución.

Como trabajo futuro se encuentran varias actividades, tales como desarrollar más métodos utilitarios para medir el porcentaje de avance de un proceso particular, el tiempo que ha tomado un proceso hasta el momento, el usuario que está ejecutando un proceso específico, entre otros. Es necesario también crear una política para determinar la prioridad de ejecución de los procesos asíncronos en *AsyncMonitor*, así como plantear otros enfoques para ejecución de métodos asíncronos y manejo de hilos.

Finalmente, es necesario realizar pruebas de concurrencia de ejecución de métodos asíncronos, además de las pruebas que ya fueron llevadas a cabo. También se contempla la realización de un instrumento de evaluación para medir la percepción de los alumnos con respecto al apoyo que se tiene con esta herramienta en la enseñanza de la comunicación asíncrona en el curso de sistemas distribuidos.

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

Bibliografía

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.