Generación de Prueba Rendimiento a Partir de Pruebas Funcionales para Sistemas Web

Federico Toledo Rodríguez[1], Matías Reina[1], Fabián Baptista[1],
Macario Polo Usaola[2], Beatriz Pérez Lamancha[3]
[1]Abstracta, Uruguay, {ftoledo, mreina, fbaptista}@abstracta.com.uy
[2]Universidad de Castilla-La Mancha, España, macario.polo@uclm.es
[3]Universidad de la República, Uruguay, bperez@fing.edu.uy

Abstract. Las pruebas de rendimiento consisten en simular carga en el sistema bajo pruebas para analizar el desempeño de la infraestructura durante la ejecución de la prueba, pudiendo encontrar cuellos de botella y oportunidades de mejora. Para la simulación se utilizan herramientas específicas, en las que se debe automatizar las acciones que generarán esa carga, esto es: las interacciones entre el usuario y el servidor. Para poder simular muchos usuarios con poca infraestructura de pruebas, se automatizan las interacciones a nivel de protocolo (en scripts), lo cual hace que la automatización sea más compleja (en cuanto al trabajo necesario para su preparación) que la automatización de pruebas funcionales, que se realiza a nivel de interfaz gráfica. Generalmente la tarea de automatización consume entre el 30% y el 50% del esfuerzo de un proyecto de pruebas de rendimiento. En este artículo presentamos la herramienta desarrollada para seguir un nuevo enfoque para generar scripts para pruebas de rendimiento a partir de scripts de pruebas funcionales. La herramienta implementada ya ha sido puesta en funcionamiento en proyectos reales, de los cuales se muestran los principales resultados que reflejan mayor flexibilidad y menor costo de automatización.

Keywords: Pruebas de Software, Pruebas de Sistemas de Información, Pruebas de Rendimiento.

1 Introducción

Dos aspectos de calidad fundamentales para reducir riesgos en la puesta en producción de un sistema son la funcionalidad y el rendimiento de un sistema.

Para verificar y mejorar la correcta funcionalidad de un sistema se recurre a pruebas a distintos niveles: pruebas unitarias, pruebas de integración, pruebas de sistema. Generalmente los proyectos de desarrollo son iterativos, contando con varias liberaciones del producto a lo largo del tiempo, ya sea a causa de la planificación de liberación de versiones, o por mantenimientos realizados luego de la puesta en producción. Es por esto que resulta necesario realizar pruebas de regresión (verificando en cada liberación que el software no tiene regresiones), y para esto generalmente se suele apoyar en herramientas de automatización de pruebas, para ejecutar estas pruebas de regresión.

Por otra parte, para las pruebas de rendimiento se utilizan también herramientas para simular la carga que generan múltiples usuarios conectados concurrentemente. Mientras se ejecuta la carga se analiza el desempeño de la aplicación en busca de cuellos de botella y oportunidades de mejora.

A continuación se profundiza sobre la automatización de pruebas funcionales (pruebas de regresión) y la automatización de pruebas de rendimiento, en especial para sistemas web, para finalmente presentar nuestra propuesta para generar pruebas de rendimiento a partir de las pruebas automatizadas funcionales.

1.1 Pruebas Funcionales Automatizadas de Sistemas Web

Una práctica muy habitual en el desarrollo de sistemas web, es la automatización de pruebas a nivel de sistema, utilizando herramientas para simular las acciones del usuario, tales como Selenium [1] y WatiN [2] (por nombrar algunos de los proyectos *opensource* más populares).

Este tipo de herramientas brindan la posibilidad de seguir un enfoque conocido como *record and playback* o capturar y reproducir. Básicamente se ejecuta manualmente un caso de prueba sobre la aplicación mientras la herramienta captura todas las acciones del usuario sobre el navegador. De esta grabación se obtiene un archivo en algún formato o lenguaje específico de la herramienta, el cual compone el script de prueba, que la misma herramienta es capaz de reproducir. De esta forma la misma prueba que ha sido grabada puede ser luego reproducida cuántas veces se quiera, realizando las validaciones que se hayan ingresado.

Un script de pruebas será entonces una secuencia de comandos que simulan las acciones de un usuario sobre una aplicación Web, que tienen como parámetros los elementos HTML sobre los que se realizó cada acción, y los valores ingresados. En la Figura 1 se puede observar un extracto de un script Selenium que accede a una aplicación (línea 1), hace clic en el link “Search” (línea 2) y por último ingresa el valor “computer” en el campo HTML de identificador “vSEARCHQUERY” (línea 3) y luego hace clic en el botón de nombre “BUTTON1”.

Command	Target	Value
open	/sampleApplication/home.aspx	
clickAndWait	link=Search	
type	id=vSEARCHQUERY	computer
click	name=BUTTON1	

Figura 1 - Script de Prueba Funcional con Selenium

Este esquema en la mayoría de las herramientas se puede complementar con un enfoque de *data-driven testing* (pruebas dirigidas por datos) haciendo que las variables del caso de prueba se tomen de una fuente de datos externa como un archivo de texto o una base de datos (llamado generalmente *datapool*). De esta forma el mismo script puede ser reproducido con distintos juegos de datos, probando así más casos con tan solo agregar más líneas de datos.

1.2 Pruebas de Rendimiento de Sistemas Web

Una prueba de rendimiento se define como una investigación técnica para determinar o validar la velocidad, escalabilidad y/o características de estabilidad de un sistema bajo prueba para analizar su desempeño en situaciones de carga [3].

Las pruebas de rendimiento son sumamente necesarias para reducir riesgos en la puesta en producción, logrando así analizar y mejorar el rendimiento de la aplicación y los servidores al estar expuestos a usuarios concurrentes. Para ello se utilizan herramientas específicas (llamadas *generadoras de carga*) para simular la acción de múltiples usuarios concurrentes. Dentro de las generadoras de carga *opensource* más populares se encuentran OpenSTA [4] y JMeter [5].

Para llevar a cabo una prueba de rendimiento generalmente se procede analizando los requisitos de performance y el escenario de carga al que se expondrá el sistema. Una vez que se seleccionan los casos de prueba estos se automatizan para su simulación en concurrencia. Una vez que se tiene lista la automatización así como la infraestructura de prueba, se pasa a ejecutar y analizar los resultados [6, 7].

Estas pruebas generalmente se realizan antes de la puesta en producción, en las etapas finales, ya que se necesita el sistema completo terminado y estable. Si se hicieran pruebas de rendimiento a un sistema que luego debe ser modificado por errores funcionales, estos cambios invalidarían los resultados obtenidos.

A diferencia de los scripts de pruebas funcionales, en estos scripts si bien se utiliza el enfoque de *record and playback*, no se graba a nivel de interfaz gráfica sino que a nivel de protocolo de comunicación. Esto es porque en una prueba funcional al reproducir se ejecuta el navegador y se simulan las acciones del usuario sobre el mismo. En una prueba de rendimiento se van a simular múltiples usuarios desde una misma máquina, por lo que no es factible abrir gran cantidad de navegadores y simular las acciones sobre ellos, ya que la máquina utilizada para generar la carga tendría problemas de rendimiento, obteniendo así una prueba inválida. Al hacerlo a nivel de protocolo se puede decir que se "ahorran recursos", ya que en el caso del protocolo HTTP lo que tendremos serán múltiples procesos que enviarán y recibirán texto por una conexión de red, y no tendrán que desplegar elementos gráficos ni ninguna otra cosa que exija mayor procesamiento.

Para preparar un script se procede en forma similar a lo explicado para los scripts de pruebas funcionales, pero esta vez la herramienta en lugar de capturar las interacciones entre el usuario y el navegador, capturará los flujos de tráfico HTTP entre el navegador y el servidor. El script resultante será una secuencia de comandos en un lenguaje proporcionado por la herramienta utilizada, en el cual se manejen los *requests* y *responses* de acuerdo al protocolo de comunicación. Esto da un script mucho más complejo de manejar que uno para pruebas funcionales. Por ejemplo, para las mismas acciones planteadas en el ejemplo de la Figura 1, que se generaban 4 líneas de script en Selenium, se genera un script de 848 líneas en OpenSTA. Esto corresponde a cada uno de los *HTTP Requests* enviados al servidor, uno para acceder a la página inicial, uno para acceder al menú de búsqueda, y uno para hacer la búsqueda. Pero además, cada *request* desencadena una serie de *requests* secundarios extra, correspondiente a las imágenes utilizadas en la página Web, archivos de estilos CSS, Javascript, y otros recursos. Incluso pueden desencadenarse redirecciones de un

objeto a otro, lo cual implica realizar más *HTTP Requests* (en respuesta a un *HTTP Request* de código *302 – redirect*).

Cada *request* (primario o secundario) se compone de un encabezado y un cuerpo de mensaje como puede verse en el ejemplo de la Figura 2. Embebidos viajan parámetros, cookies, variables de sesión, y todo tipo de elementos que se utilicen en la comunicación con el servidor. El ejemplo de la figura corresponde al *HTTP Request* primario del último paso del caso de prueba, por lo que se puede ver que se envía el valor "computer" en el parámetro "vSEARCHQUERY" (en el recuadro rojo).

Una vez que se graba el script, es necesario realizar una serie de ajustes sobre estos elementos para que quede reproducible. Estos scripts serán ejecutados por usuarios concurrentes, y por ejemplo, no tiene sentido que todos los usuarios utilicen el mismo nombre de usuario y clave para conectarse, o que todos los usuarios hagan la misma búsqueda (ya que en ese caso la aplicación funcionaría mejor que de usar diferentes valores, dado que influirían los cachés, tanto a nivel de base de datos como a nivel de servidor de aplicaciones Web). El costo de este tipo de ajustes dependerá de la herramienta utilizada y de la aplicación bajo pruebas. En ocasiones es necesario ajustar cookies o variables, pues las obtenidas al grabar dejan de ser válidas, debiendo ser únicas por usuario. Se deberán ajustar parámetros, tanto del encabezado como del cuerpo del mensaje, etc.

Figura 2 - Script de Prueba de Rendimiento con OpenSTA

En base a nuestra experiencia en más de 8 años realizando pruebas de rendimiento, la construcción de estos scripts ocupa entre el 30% y el 50% del total de esfuerzo invertido en las pruebas de rendimiento. Por otra parte, el mantenimiento de estos scripts suele ser tan complejo que se prefiere rehacer un script de cero en lugar de ajustarlo. De esta forma el proceso se vuelve poco flexible en la práctica. Las pruebas generalmente identificarán oportunidades de mejora sobre el sistema, con lo cual se sugerirán cambios a realizar, pero por otra parte si se modifica el sistema es probable que se deban rehacer los scripts. ¿Cómo se verifica que el cambio realizado ha dado buenos resultados?

1.3 Enfoque Propuesto

Dado que el enfoque de automatizar pruebas a nivel de interfaz gráfica es más simple que a nivel de protocolo, y más fácil de mantener, lo que se plantea es aprovechar las pruebas automatizadas funcionales para generar automáticamente scripts para pruebas de rendimiento.

Básicamente la propuesta consta en ejecutar los scripts de pruebas funcionales y capturar el tráfico HTTP generado durante esa ejecución. Luego, ese tráfico capturado se analiza para obtener un modelo, el cual puede ser transformado en código para ser utilizado en la herramienta generadora de carga deseada.

El resto del artículo se organiza de la siguiente manera: en la sección 2 se presenta la propuesta, que luego se valida con el caso de estudio en la sección 3, y luego de presentar algunos trabajos relacionados en la sección 4 se mencionan las conclusiones y líneas de trabajo futuro en la sección 5.

2 Generación Automática de Pruebas de Rendimiento

Las herramientas de generación de carga generan un script de pruebas a partir de la ejecución manual de un caso de prueba de un usuario, capturando el tráfico HTTP producido, y transcribiéndolo luego en un script que es luego ejecutable en el mismo entorno de la herramienta. En nuestro enfoque realizaremos algo similar, pero en lugar de pedir al usuario que ejecute los casos de prueba manualmente, simplemente los tendrá que seleccionar a partir de los scripts de pruebas funcionales que ya tenga. Tal como muestra la Figura 3, a partir de la ejecución de estos scripts se obtendrá un modelo del tráfico HTTP capturado. Este modelo por último es utilizado como entrada para el proceso que genera el código en la herramienta de generación de carga seleccionada.

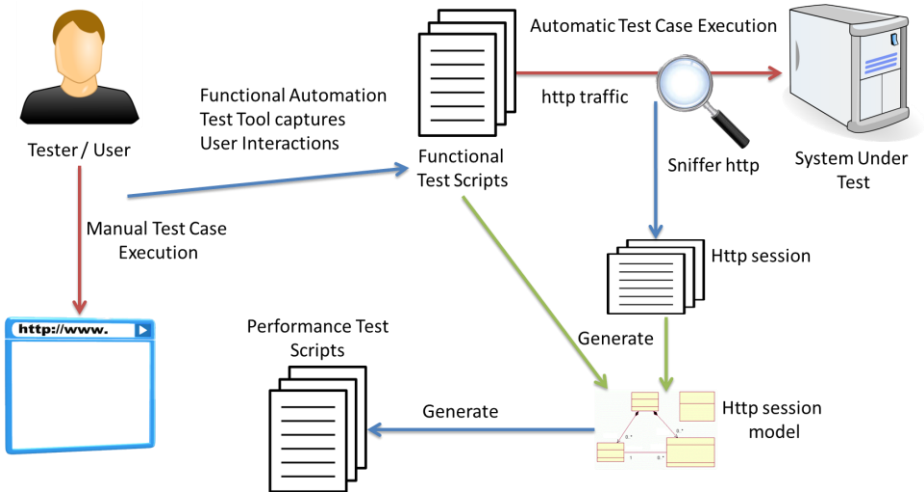

Figura 3 - Propuesta de Generación de Scripts para Pruebas de Rendimiento

La herramienta implementada ejecuta scripts Selenium y WatiN. Durante la ejecución de estas pruebas captura el tráfico HTTP entre el navegador y la aplicación bajo pruebas utilizando un *sniffer web* (herramienta capaz de capturar tráfico de red de las peticiones web) llamado Fiddler [8], específico para HTTP. Con esta información almacenada se construye un modelo del tráfico, del cual se generan los scripts para ser ejecutados con OpenSTA.

La Figura 4 muestra los principales elementos del modelo de tráfico que es útil para generar los scripts de pruebas de rendimiento. Este modelo se genera a partir del script de pruebas funcionales y de la información obtenida con el sniffer, esto es, todos los *requests* y *responses* HTTP que se generaron al ejecutar el correspondiente script de pruebas funcionales.

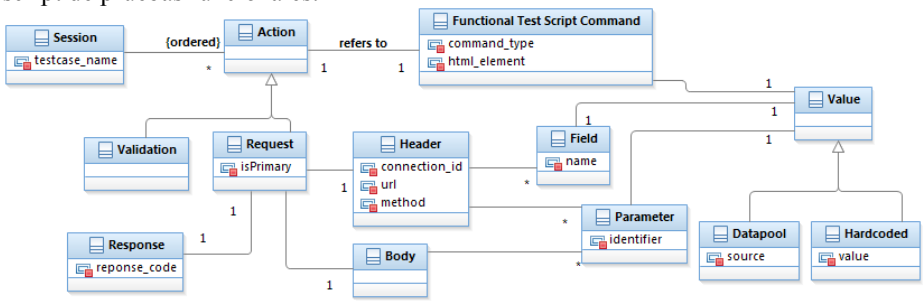

Figura 4 - Modelo de Sesión HTTP

Básicamente, el modelo representa información sobre la sesión HTTP generada a partir de la ejecución del caso de prueba, por lo que se compone de una secuencia ordenada de acciones, que pueden ser invocaciones HTTP sobre la aplicación (*request*), o validaciones para verificar si la respuesta fue correcta (*validation*). Cada *HTTP Request* se compone de un encabezado (*header*) y cuerpo del mensaje (*body*). Ambas partes del mensaje se componen de parámetros con sus respectivos valores. El encabezado en particular, tiene un conjunto de campos con su valor asociado, entre los que se encuentran las cookies, datos de sesión, etc. Los valores que toman los parámetros pueden ser valores fijos (*hardcoded*) o pueden ser tomados de una fuente de datos externa (*datapool*). Es importante guardar la referencia entre cada invocación HTTP y su respuesta (*response*), así como con el comando del script funcional que la originó (*functional test script command*).

Con este modelo se genera código para la herramienta de generación de carga utilizada. En particular, genera código para OpenSTA. Para ello se sigue un enfoque similar al de las herramientas de transformación de modelo-a-texto [9], donde se cuenta con plantillas o *templates* de código para cada elemento del modelo. En la Tabla 1 se presentan algunos ejemplos de estos *templates*, donde se puede ver en la primera fila la estructura general de un script, la cual es generada para cada caso de prueba; en la segunda fila se presenta el *template* para verificar que un texto aparece en una respuesta (la función "AssertText" es parte de una biblioteca de funcionalidades estándar que también se genera automáticamente); y la tercera fila corresponde a un *HTTP Request*, donde se leen las cookies que devuelve el servidor, y se cargan en variables para poder enviarla en los siguientes *requests*, de acuerdo al protocolo HTTP.

Tabla 1 - Templates para Generación de Código de Scripts

```
[template public openSTA_Script(s: Session)]
[s.testcase_name/]
[/template]
[template private openSTA_Script_File(s: Session)]
[s.openSTA_Script()/].scl
[/template]
[template public generateScript(s: Session)]
[file (s.openSTA_Script_File(), false, 'UTF-8')]
Environment
        Description ""
        Mode    HTTP
        Wait    UNIT MILLISECONDS
Definitions
        ! Standard Defines
        Include "RESPONSE_CODES.INC"
        Include "GLOBAL_VARIABLES.INC"
        CHARACTER*512   USER_AGENT
        Integer         USE_PAGE_TIMERS
        CHARACTER*256   MESSAGE
        Timer T_TestCase
        [s.timerDefinitions()/]
        [s.variableDeclarations()/]
        CONSTANT DEFAULT_HEADERS = "Host:  [s.getBaseURL()/]
        User-Agent: Mozilla/4.0"
Code
        !Read in the default browser user agent field
        Entry USER_AGENT,USE_PAGE_TIMERS
        Start Timer T_TestCase
        [s.processActions()/]
        End Timer T_TestCase
Exit
ERR_LABEL:
        If (MESSAGE <> "") Then
                Report MESSAGE
        Endif
Exit
[/file]
[/template]
```

```
[template public processValidation(v: Validation)]
Set buffer=''
Set expectedResponse = [v.expectedValue/]
Load Response_Info Body ON [v.request.header.connection_id/] Into buffer
Include "AssertText.inc"
[/template]
```

```
[template public processRequest(r: Request)]
Start Timer [r.name/]
[if ([r.isPrimary/])]PRIMARY [/if] [r.header.method/] URI [r.header.url/]
HTTP/1.1" ON [r.header.connection_id/] &
        HEADER DEFAULT_HEADERS &
        ,WITH [r.header.processFields()/]}
        [r.processBody()/]
[r.response.processLoadCookies()/]
End Timer [r.name/]
[/template]
```

Como se comentó antes, luego de grabar un script de pruebas de rendimiento, es necesario realizar una serie de ajustes. Muchos de ellos se vuelven tareas muy repetitivas. Para este tipo de cosas el generador de scripts lo realiza en forma automática aprovechando su estructura de *templates*. Entre ellas se destacan:

- Agregar *timers* para cada acción del usuario, que permiten medir los tiempos de respuesta al momento de ejecutar las pruebas. Esto se puede realizar automáticamente pues se tienen en cuenta el tipo de acciones realizadas en el script de pruebas funcionales, y los *HTTP Requests* desencadenados para cada una.
- Parametrizar las variables del caso de prueba. Si la prueba a nivel funcional estaba diseñada para tomar valores de una fuente de datos, se genera el mecanismo de data-driven testing a nivel de la prueba de rendimiento automáticamente.
- Agrega al script de rendimiento las validaciones realizadas en el script de pruebas funcionales. Generalmente se hacen verificaciones sobre las respuestas, buscando que los valores retornados por el sistema sean correctos. Agregar estas validaciones a nivel de protocolo es más complejo que a nivel de interfaz gráfica, ya que se debe buscar en el código HTML de la respuesta. Estas validaciones son agregadas automáticamente.
- Estructura el código y lo modulariza en distintos archivos, colaborando con la legibilidad del script de pruebas. Los *requests* secundarios (imágenes, CSS, Javascript) son puestos en scripts separados (invocados desde el script principal) para reducir el tamaño del código y dejar un script más fácil de entender y modificar en caso que se desee. Además, si un script de pruebas funcionales está modularizado, también se genera el script de prueba de rendimiento con la misma modularización.
- Agrega automáticamente los comandos para la autenticación. En OpenSTA esto es necesario agregarlo manualmente después de la grabación. Si está resuelto en el script de pruebas funcionales se genera automáticamente para el script de pruebas de rendimiento.

De esta forma se consiguen scripts de mejor calidad que los que genera el *recorder* de OpenSTA al grabar un caso de prueba en forma manual. A medida que se ha utilizado la herramienta se han ido introduciendo más ajustes automáticos, que tienen como ventaja que se hacen automáticamente, y evita la posibilidad de errores del tester al preparar el script de pruebas de rendimiento.

Una vez que se cuentan con los scripts, se puede pasar a la parte más importante (e interesante y beneficiosa) de las pruebas de rendimiento, que es la ejecución y análisis de resultados.

3 Caso de Estudio

La herramienta se ha utilizado en cinco proyectos distintos, sobre sistemas Web de diversa índole, obteniendo muy buenos resultados en todos ellos.

En la Tabla 2 se puede observar la cantidad de scripts que se generaron para cada proyecto, y la cantidad de usuarios concurrentes accediendo a un sistema que se

simularon en cada caso. Observar que los sistemas son de diversos dominios, y distintas plataformas.

Tabla 2 - Uso de la Herramienta en Pruebas de Rendimiento

Proyecto	Sistema Bajo Pruebas	# Scripts	# Usuarios Simulados
P1 – Sistema de Gestión de Recursos Humanos	Base de datos en AS400, Sistema Web Java en Websphere	14	317
P2 – Sistema de Gestión de Productores	Base de datos en AS400, Sistema Web C# en Internet Information Services	5	55
P3 – Sistema de Gestión de Tribunales	Sistema Web Java en Tomcat con base de datos Oracle	5	144
P4 – Sistema de Subastas	Sistema Web Java en Tomcat con base de datos MySQL	1	2000
P5 – Sistema de Logísitica	Sistema Web Java en Weblogic con base de datos Oracle	9	117

Vale aclarar que casos en los que hay pocos scripts, como en P4 que sólo hay uno, se definieron de este modo ya que la mayor parte de la carga está concentrada en pocas funcionalidades, tal vez con distintas variantes representadas internamente en el script. Además, todos los scripts han sido ejecutados con diversos juegos de datos.

En algunos proyectos se probaron sistemas desarrollados con herramientas de desarrollo dirigido por modelos (en particular con una herramienta llamada GeneXus [10]), donde se genera código a partir de los modelos diseñados en alto nivel de abstracción. Esto plantea una complejidad especial al escenario, ya que pequeñas modificaciones al modelo de desarrollo pueden significar en grandes modificaciones en el flujo HTTP de la aplicación generada. El procedimiento seguido fue el mismo que en otro tipo de aplicaciones: se ajustaron los scripts de pruebas funcionales, y se regeneraron los scripts de pruebas de rendimiento. En este esquema de trabajo donde el sistema bajo pruebas tiene grandes variaciones es donde mayores beneficios reportó el enfoque propuesto, ya que fue necesario regenerar los scripts de pruebas varias veces, y de tener que hacerlo manualmente el esfuerzo se hubiera multiplicado.

En un proyecto en el que se comenzaría a trabajar directamente realizando pruebas de rendimiento, se utilizó la herramienta, por lo cual se desarrollaron scripts de pruebas funcionales que resultan útiles para la ejecución de las pruebas de regresión. Una vez que finalizó el proyecto, se comenzó a gestionar el ambiente de pruebas de regresión incluyendo las pruebas diseñadas para las pruebas de rendimiento. Entonces, de cierta forma el control de calidad de rendimiento favoreció el control de calidad de funcionalidad.

4 Trabajos Relacionados

Existen herramientas que, para facilitar la construcción y el mantenimiento de las pruebas de rendimiento, trabajan a nivel de interfaz gráfica, utilizando scripts de Selenium para pruebas de rendimiento. El problema es que para poder ejecutar esos scripts, con las cantidades de usuarios típicas de una pruebas de rendimiento, lo hacen desde el Cloud, o necesitando una infraestructura de pruebas muy grande. Algunos ejemplos son Scaleborn (www.scaleborn.com) y Test Maker (www.pushtotest.com). Con nuestro enfoque, en cambio, se puede conservar reducida la infraestructura necesaria, siendo una alternativa más económica, con los mismos resultados.

Por otra parte, la son pocas las propuestas que conocemos para generar pruebas de rendimiento en forma automática. Existen varias que proponen diseñar modelos como base para la generación de pruebas, como es el caso de la propuesta de García-Domínguez et al. [11], la cual se centra en pruebas de rendimiento para flujos de trabajo (*workflows*), invocando Web Services. Otros proponen utilizar modelos UML estereotipados, tales como [12–14], o incluso extienden una herramienta de diseño UML para generar todo un set de artefactos de pruebas de rendimiento a partir de los diagramas modelados [15]. La desventaja en estos enfoques está en el esfuerzo extra necesario para diseñar los artefactos de entrada necesarios para el generador de scripts. Por último destacar el trabajo realizado por de Sousa et al. [16] donde muestran un enfoque similar al propuesto en este artículo, aprovechando scripts de pruebas funcionales para generar scripts de pruebas de rendimiento. El problema de este enfoque es que al no considerar un flujo HTTP de la ejecución del caso de prueba y sólo tener como entrada el script de prueba funcional, es imposible considerar los *requests* secundarios o redirecciones que esté realizando la aplicación, con lo cual las pruebas generadas no son realistas, no simulan la actividad de los usuarios fielmente.

5 Conclusiones y Trabajo Futuro

Las pruebas de rendimiento son necesarias para reducir riesgos en la puesta en producción de un sistema, pero al ser tan costosas y demandantes de recursos, estas suelen llegar tarde, o en mala forma. La tarea que más recursos demanda es la automatización de las funcionalidades que se desean probar, robando parte del tiempo disponible a la ejecución de las pruebas que es la que realmente da los resultados.

Por este motivo, se presenta en este artículo una propuesta para generar scripts para pruebas de rendimiento aprovechando los scripts para pruebas funcionales. Esto no solo da mayor flexibilidad a la hora de ajustar las pruebas a los cambios y mejoras de la aplicación (que se realizan necesariamente a lo largo del proyecto de pruebas) sino también que permite realizar la tarea con calidad y en menor tiempo.

La herramienta implementada con este enfoque se está utilizando en diversos proyectos para realizar las pruebas de rendimiento, con lo que se están demostrando los beneficios del enfoque propuesto.

A futuro se pretende extender la generación de scripts de pruebas de rendimiento a otras herramientas, tales como JMeter, que tiene como ventaja ante OpenSTA que da soporte a más protocolos, y se puede de esta forma probar un sistema al cual se

accede por diversas interfaces (HTTP, SOAP, Rest, FTP) gestionando la prueba desde una sola herramienta.

Agradecimientos. Este trabajo ha sido parcialmente financiado por la Agencia Nacional de Investigación e Innovación (ANII, Uruguay) y por el proyecto GEODAS-BC (Ministerio de Economía y Competitividad y Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01). Agradecimiento especial también al equipo de Abstracta.

Referencias

1. Huggins, J.: Selenium, http://seleniumhq.org/.
2. Jeroen van Menen: WatiN, http://watin.org/.
3. Meier, J., Farre, C., Bansode, P., Barber, S., Rea, D.: Performance testing guidance for web applications: patterns & practices. Microsoft Press (2007).
4. Cyrano: OpenSTA, http://opensta.org/.
5. Apache: JMeter, http://jmeter.apache.org/.
6. Gustavo Vázquez, Matías Reina, Federico Toledo, Simón de Uvarow, Edgardo Greisin, Horacio López: Metodología de Pruebas de Performance. Presented at the XX Encuentro Chileno de Computación - Jornadas Chilenas de Computación , Punta Arenas, Chile November 10 (2008).
7. Barber, S.: User Experience, not Metrics. (2001).
8. Eric Lawrence: Fiddler, http://www.fiddler2.com.
9. OMG: MOF Model to Text Transformation Language (MOFM2T), 1.0. (2008).
10. Artech: GeneXus, http://www.genexus.com.
11. García-Domínguez, A., Medina-Bulo, I., Marcos-Bárcena, M.: Performance Test Case Generation for Java and WSDL-based Web Services from MARTE. International Journal On Advances in Internet Technology. 5, 173–185 (2012).
12. Garousi, V., Briand, L.C., Labiche, Y.: Traffic-aware stress testing of distributed systems based on UML models. Proceedings of the 28th international conference on Software engineering. pp. 391–400. ACM, New York, NY, USA (2006).
13. Shams, M., Krishnamurthy, D., Far, B.: A model-based approach for testing the performance of web applications. Proceedings of the 3rd international workshop on Software quality assurance. pp. 54–61. ACM, New York, NY, USA (2006).
14. Silveira, M.B. da, Rodrigues, E. de M., Zorzo, A.F., Costa, L.T., Vieira, H.V., Oliveira, F.M. de: Generation of Scripts for Performance Testing Based on UML Models. SEKE. pp. 258–263 (2011).
15. Cai, Y., Grundy, J., Hosking, J.: Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool. In Proceedings of the 2004 IEEE International Conference on Automated Software Engineering. pp. 36–45. ASE (2004).
16. Santos, I. de S., Santos, A.R., Neto, P. de A. dos S.: Reusing Functional Testing in order to Decrease Performance and Stress Testing Costs. SEKE. pp. 470–474 (2011).