Una Revisión de los Métodos de Pruebas para Aplicaciones Web

Anibal Minaya Cubillas[1] - David Mauricio[2]

Universidad Nacional Mayor de San Marcos,
Facultad de Ingeniería de Sistemas e Informática
Av. Germán Amézaga s/n, Ciudad Universitaria, Lima 01, Lima, Perú

[1]aminayac@gmail.com, [2]dms_research@yahoo.com

RESUMEN

El gran uso y aceptación de Internet en las últimas décadas ha ocasionado un gran crecimiento en la construcción de aplicaciones basadas en Web, las cuales han evolucionado desde simples sitios Web con aplicaciones formadas por páginas estáticas hasta complejas aplicaciones distribuidas con hardware y software heterogéneo. Estas aplicaciones adicionalmente tienen requerimientos cada vez más estrictos de confiabilidad, facilidad de uso, interoperabilidad con sistemas existentes, rendimiento y seguridad. Las pruebas de software son un proceso dificultoso y lo son más aún cuando se trata de aplicaciones basadas en Web, debido a las peculiaridades de tales aplicaciones. Estas pruebas pueden ser de varios tipos, ya sea si están encaminadas a probar las características funcionales (pruebas funcionales) o no funcionales del software (pruebas de rendimiento, carga, seguridad, etc.). El presente trabajo hace una revisión de los métodos de prueba de aplicaciones web, tanto de pruebas de caja negra, blanca y gris, haciendo mayor énfasis en estas últimas, específicamente la basada en la generación de casos de prueba a partir de la información de sesiones de usuario recolectada en el servidor web.

Palabras claves: pruebas de aplicaciones web, sesiones de usuario, casos de prueba.

ABSTRACT

The extensive use and acceptance of the Internet in recent decades has caused a great increase in building Web-based applications, which have evolved from simple Web sites with applications consisting of static pages to complex distributed applications with heterogeneous hardware and software. These additional applications are increasingly stringent requirements for reliability, usability, interoperability with existing systems, performance and security. The software testing is a difficult process and even more when it comes to Web-based applications, due to the peculiarities of such applications. Such evidence may take various forms, whether they are designed to test the functional characteristics (tests) or non-functional software (performance testing, load, security, etc.). This paper makes a review of methods for testing web applications, both black boxes testing, white and gray, with greater emphasis on the latter, specifically based on the generation of test cases from the information sessions collected in the user web server.

Keywords: web applications testing, user session data, test-suite

1. INTRODUCCIÓN

En general, el proceso de prueba de aplicaciones es un proceso complejo y con tendencia a cometer errores en su ejecución. Este proceso tiene como finalidad verificar y validar la calidad del software que se está desarrollando. Los distintos tipos de pruebas tienen como objetivo identificar algún error o discrepancia entre lo que hace actualmente el software contra las especificaciones del software (funcionales o no funcionales). Sin embargo, a pesar de su importancia, las pruebas son consideradas como actividades tediosas y dificultosas y muchas de ellas son ignoradas. Las pruebas funcionales (pruebas que validan que el software realice lo que dice la especificación del usuario) son realizadas la mayoría de las veces en forma manual ocasionando demora en el proceso de pruebas, ya que la construcción de los casos de pruebas puede ser bastante laboriosa.

Aunque existen actualmente muchos enfoques y herramientas que apoyan en el proceso de pruebas, éstos se orientan principalmente a probar características no funcionales de las aplicaciones (principalmente el rendimiento, carga, detección de links rotos y otros análisis estáticos) y no sus características funcionales. [Elbaum 2003].

La investigación actual de generación de casos de prueba funcionales para aplicaciones Web actualmente considera varios enfoques. Uno de estos enfoques consiste en generar casos de prueba en base a información almacenada en archivos de bitácora (archivos log) que contienen todas las invocaciones que han realizado los usuarios a los objetos pertenecientes a una aplicación y sus respectivos parámetros (esta información es conocida como datos de sesión de usuario o "user session data" y son almacenados por el servidor Web que contiene a la aplicación). Este tipo de enfoque es principalmente ventajoso para pruebas de regresión en aplicaciones Web, ya que éstas evolucionan o son modificadas rápidamente [Sant 2005]. El problema que se presenta con este enfoque es la optimización de la construcción de los casos de prueba, ya que al requerirse mayor información en los archivos de bitácora (para tener mayor probabilidad de encontrar inconsistencias o errores) el manejo de los archivos de bitácora y la generación de los casos de prueba se hace inmanejable y costoso [Sprenkle 2006]. Otros enfoques se basan en técnicas de pruebas que se aplican a aplicaciones tradicionales, siendo el problema con estas técnicas la manera distinta en que están construidas las aplicaciones Web (software y hardware distribuidos) y su rápido cambio debido a modificaciones [Sant 2005].

Según lo mencionado, se requieren de métodos de prueba que no requieran un gran costo, tiempo y esfuerzo. Si alguno de los métodos existentes pudiera ser automatizado, ayudará a que el proceso de prueba sea menos pesado de lo que ahora resulta para los equipos de pruebas. En el presente trabajo se hace una revisión de los métodos de prueba de aplicaciones web, tanto de pruebas de caja negra, blanca y gris, haciendo mayor énfasis en estas últimas, específicamente la basada en la generación de casos de prueba a partir de la información de sesiones de usuario recolectada en el servidor web.

El resto del trabajo está organizado como sigue. En la sección 2 se trata sobre las pruebas de aplicaciones web y sus particularidades respecto a aplicaciones tradicionales. La sección 3 trata sobre las pruebas no funcionales de aplicaciones web. La sección 4 se centra en las pruebas funcionales y los distintos tipos de estas pruebas, así como detalla las pruebas basadas en sesiones de usuario. La sección 5 se centra en las técnicas de reducción de casos de prueba. Finalmente, la sección 6 muestra las conclusiones del trabajo.

2. PRUEBAS DE APLICACIONES WEB

2.1 Aplicaciones WEB

Una aplicación Web es una aplicación que se corre sobre un servidor de aplicaciones, está disponible al usuario a través de la red y el usuario la ejecuta a través de un navegador. Como se muestra en la Figura 1, los pasos en el flujo de ejecución de una aplicación Web son los siguientes:

(1) El usuario hace un requerimiento a la aplicación web usando un navegador. Vía un navegador Web (Mozilla Firefox, Microsoft Internet Explorer o Apple Safari) un usuario solicita un recurso de la aplicación web identificado por una URL (Universal Resource Locator) utilizando HTTP (Hipertext Transfer

Protocol). La solicitud HTTP incluye el tipo de solicitud (GET ó POST), la ubicación del recurso solicitado y data opcional, tales como variables y sus valores dentro de un formulario.

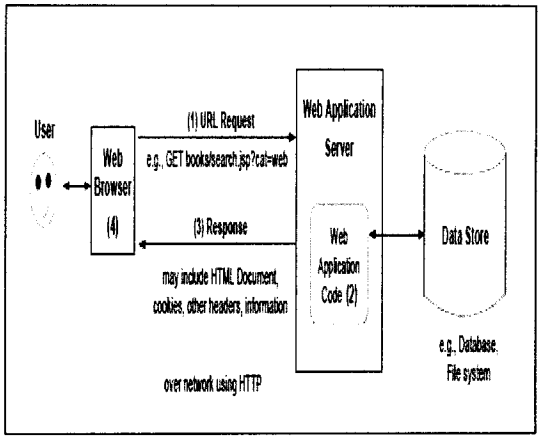

Fig. 1. Arquitectura General de una aplicación Web [Sprenkle 2007]

(2) La aplicación Web genera la adecuada respuesta a la solicitud del usuario. El usuario puede solicitar uno de los recursos estáticos o dinámicos de la aplicación Web. Los recursos estáticos son los que la aplicación no genera dinámicamente (archivos Javascript, imágenes, archivos CSS, o archivos HTML.). Para los recursos dinámicos, la aplicación Web genera dinámicamente una respuesta basada en la solicitud del usuario y en el estado actual de la aplicación. Para generar esta respuesta, la aplicación puede acceder a una fuente de datos (base de datos relacional, archivos de datos, etc.)

(3) La aplicación web envía la respuesta al solicitante. La aplicación web empaqueta el recurso dinámico o estático en una respuesta HTTP y lo envía al solicitante (navegador web)

(4) El navegador muestra la respuesta, de tal forma que el usuario pueda verla. El navegador es el responsable de mostrar los documentos HTML a los usuarios [Sprenkle 2007]

2.2 Desafíos en las pruebas de aplicaciones web

Más allá de los desafíos para garantizar la fiabilidad de las aplicaciones tradicionales, las aplicaciones web introducen desafíos de pruebas adicionales. Tras el despliegue, las aplicaciones web con frecuencia se someten a mantenimiento para corregir errores, agregar funcionalidad y mejorar el rendimiento [Offutt 2002]. Además, las aplicaciones web tienen un corto período de comercialización, cambios frecuentes en el uso del cliente, persistencia y, posiblemente, un gran sistema distribuido y compuesto por diversos componentes (incluidos variados navegadores clientes y bibliotecas de terceros), y clientes que acceden a una aplicación simultáneamente las 24 horas del día. Con la prevalencia de uso de las aplicaciones web para realizar actividades diarias en todo tipo de negocios, incluso una pérdida de funcionalidad parcial puede costar a las empresas millones de dólares por hora. Por lo tanto, una completa y eficaz prueba de aplicaciones web de una forma que imita la interacción de los usuarios es fundamental para garantizar la funcionalidad de la aplicación y que no se ha visto afectada por los cambios y mantenimiento y para reducir el tiempo de inactividad para solucionar sus problemas.

2.3 Tipos de pruebas para aplicaciones web

Tomando como perspectiva principal los requerimientos de la aplicación Web, según [Di Lucca 2006] existen dos tipos de pruebas: no funcionales y funcionales. Las primeras están orientadas a validar características de performance, usabilidad, portabilidad, etc. mientras que las pruebas funcionales permiten validar la correcta implementación de la funcionalidad del software según lo solicitado por el usuario

3. PRUEBAS DE REQUERIMIENTOS NO FUNCIONALES

Existen diferentes tipos de pruebas no funcionales que son aplicadas a aplicaciones Web. Cada una de estas pruebas tiene que ser verificadas contra el requerimiento no funcional establecido, para determinar si pasó o no la prueba. Las pruebas no funcionales son:

Pruebas de rendimiento: Son utilizadas para comprobar características relacionadas al rendimiento (performance), tales como tiempo de respuesta y disponibilidad de servicio. Estas pruebas son realizadas simulando cientos o

miles de usuarios usando la aplicación sobre un determinado intervalo de tiempo.

Pruebas de carga: Son utilizadas para evaluar el tiempo necesitado para realizar varias tareas y funciones bajo condiciones predefinidas.

Pruebas de stress: Son utilizadas para comprobar la respuesta de la aplicación cerca o más allá de los límites de sus requerimientos especificados (al contrario de los dos tipos de pruebas anteriores que consideran un uso regular de la aplicación).

Pruebas de compatibilidad: Estas pruebas están orientadas a la comprobar si la aplicación funciona en distintas plataformas de servidores Web, navegadores clientes, etc.

Pruebas de facilidad de uso: Estas pruebas están orientadas a comprobar la facilidad de uso de la aplicación por parte del usuario. Consiste principalmente en probar la interfaz gráfica del usuario validando que se muestren correctamente los gráficos, claridad de los mensajes, etc.

Pruebas de accesibilidad: Estas pruebas son utilizadas para validar el funcionamiento del software en entornos limitados de hardware y software (por ejemplo, navegadores que tienen deshabilitados los scripts y gráficos) o que puedan ser usados por personas con discapacidades (por ejemplo personas ciegas).

Pruebas de seguridad: Estas pruebas están orientadas a comprobar las defensas de la aplicación, de tal forma que solamente los usuarios autorizados puedan acceder a ella. Adicionalmente, en estas pruebas se verifica que los usuarios que pueden acceder a la aplicación sólo tengan acceso a las opciones especificadas a sus roles.

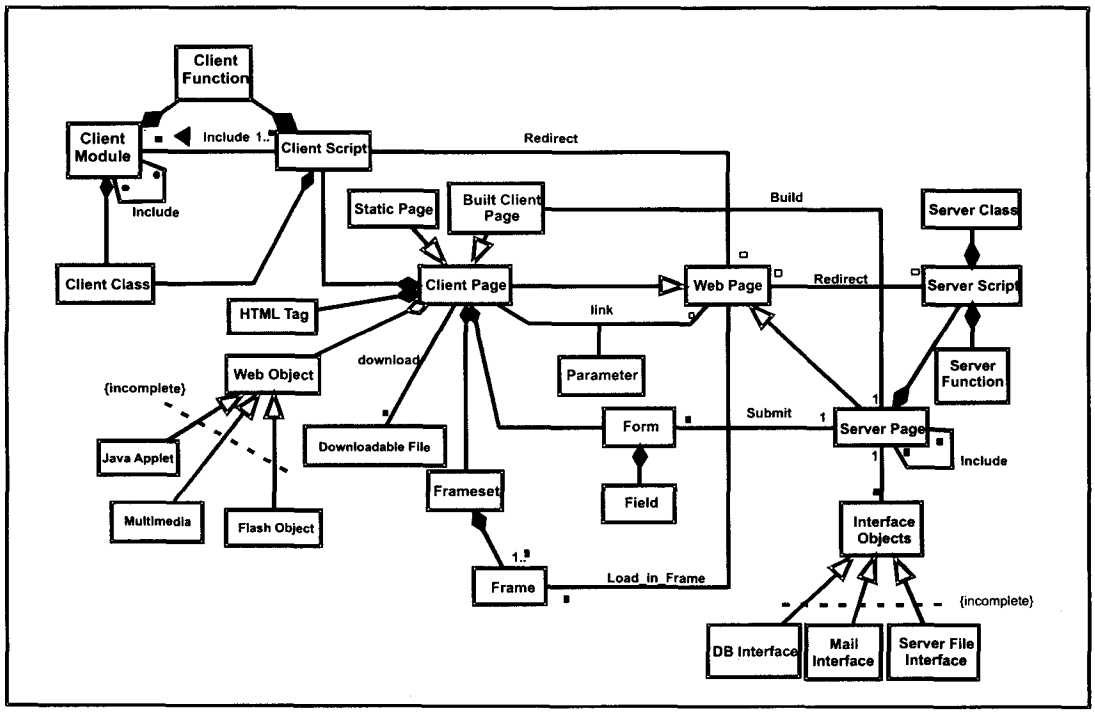

Fig. 2. Meta-Modelo para una aplicación Web [Di Lucca 2006]

4. PRUEBAS DE REQUERIMIENTOS FUNCIONALES

Estas pruebas están orientadas a hallar errores debidos a una mala implementación de los requerimientos funcionales de la aplicación. Aunque estos enfoques de pruebas se encuentran no muy desarrollados [Hao 2006], éstas aun se basan en aspectos que son usados en las pruebas de aplicaciones

tradicionales. Según [Di Lucca 2006], estos aspectos son:

- Modelos para pruebas
- Niveles de pruebas
- Estrategias para pruebas

4.1 Modelos para pruebas

Dependiendo de los ítems que se desean probar, los modelos son necesarios para representar los conceptos y relaciones acerca de estos ítems. Basados en estos modelos se construyen los casos de prueba necesarios. Existen varios meta-modelos planteados en la literatura existente, uno de los cuales es el mostrado en la figura 2.

4.2 Niveles de pruebas

Pruebas unitarias. Distintos tipos de pruebas unitarias pueden ser deducidas del meta-modelo mostrado en la figura anterior, sin embargo, la unidad básica de pruebas unitarias de aplicaciones Web es la página Web (Web Page en el meta-modelo).

Pruebas de integración. En estas pruebas, varias páginas relacionadas son probadas para validar su funcionalidad. Como en el caso anterior, basando el criterio en el meta-modelo anterior, identificamos conjuntos de páginas relacionadas con criterios como redirect, submit, etc.

Pruebas de sistemas. Estas pruebas validarán la funcionalidad de toda la aplicación Web.

4.3 Estrategias Para Pruebas

Las estrategias para pruebas definen los enfoques para diseñar los casos de prueba. Para aplicaciones Web existen las siguientes estrategias: caja blanca, caja negra y caja gris.

Caja blanca

Las estrategias de caja blanca diseñan los casos de prueba sobre la base de una representación de código del componente que se está probando y de un modelo de cobertura que especifica qué partes de la representación deben ser probadas por el caso de prueba.

Existen dos técnicas propuestas para pruebas de caja blanca de aplicaciones Web: la propuesta por Liu et al. [Liu 2000] y la propuesta por Ricca y Tonella, [Ricca 2001].

Las pruebas propuesta por Liu et al. [Liu 2000], se aplica a aplicaciones Web implementadas en HTML y XML, así como otros tipos de componentes ejecutables (Java Applets, Controles ActiveX, etc.). Este enfoque es basado en el Modelo de Pruebas de Aplicaciones Web (WATM: Web Application Test Model), el cual incluye un modelo de objetos y un modelo de estructuras.

El modelo de objetos representa los elementos heterogéneos de una aplicación Web y cómo ellos están conectados. El modelo incluye tres tipos de objetos (páginas cliente, páginas de servidor y componentes), así como siete tipos de relación entre ellos (herencia, agregación, asociación, request, response, navegación y redirección). El modelo de estructuras utiliza cuatro tipos de gráficos para capturar varios tipos de información de flujo de data (data flow): El Gráfico de Flujo de Control (CFG) para una función individual, el Gráfico de Flujo de Control InterProcedural (ICFG), que integra más de una función, el Gráfico de Flujo de Control de Objetos (OCFG) y finalmente el Gráfico de Flujo de Control Compuesto (CCFG).

Este método de prueba tiene el mérito de aplicar los enfoques de pruebas de flujo de datos (data flow), algo que hasta ese momento sólo se usaba en aplicaciones tradicionales. Sin embargo, mayor investigación es requerida en este campo.

La segunda técnica, propuesta por Ricca y Tonella [Ricca 2001], principalmente se aplica a aplicaciones web estáticas. Esta técnica se basa en el modelo denominado Modelo Navegacional, el cual se centra en las páginas HTML y los links que permiten la navegación entre éstas (figura 3).

Un caso de prueba para la aplicación Web consiste de una secuencia de páginas que serán accedidas más los valores de entrada que serán suministrados a las páginas que contengan formularios. Se han propuesto varios criterios de cobertura para generar los casos de prueba, entre ellos Cobertura de caminos (Path Coverage), el cual requiere que todos los caminos en el modelo de la aplicación web sean atravesados por algún caso de prueba; Cobertura de ramas (Branch Coverage), el cual

requiere que todas las ramas en el modelo sean atravesadas por algún caso de prueba y Cobertura de nodos (Node Coverage), el cual requiere que todos los nodos en el modelo sean atravesados por algún caso de prueba.

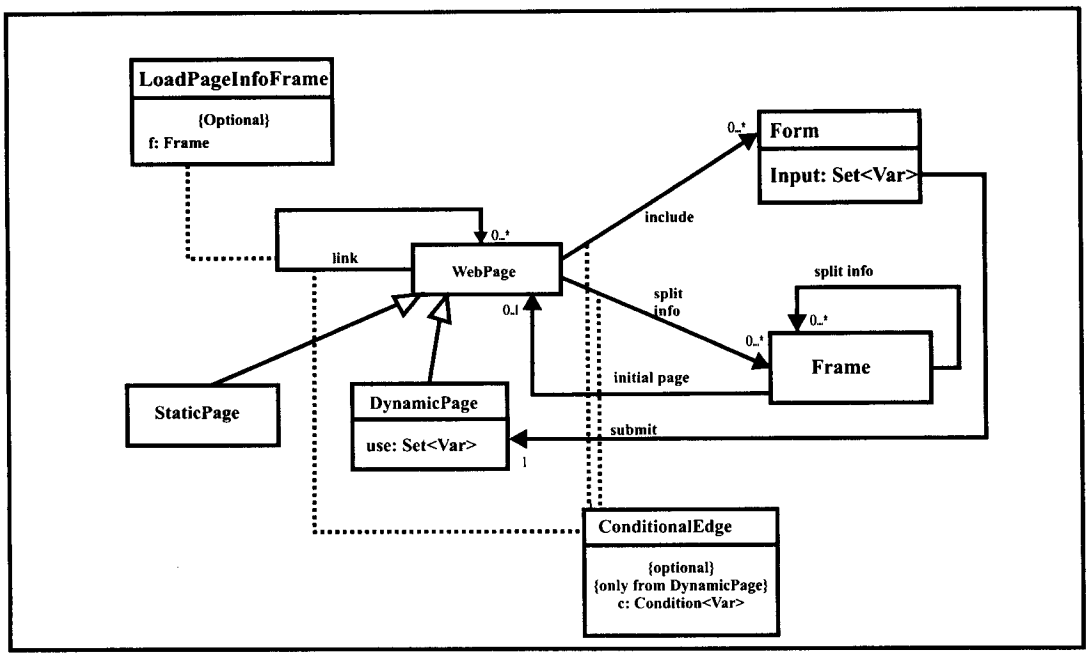

Fig. 3. Modelo de una aplicación Web [Ricca 2001]

Un caso de prueba para la aplicación Web consiste de una secuencia de páginas que serán accedidas más los valores de entrada que serán suministrados a las páginas que contengan formularios. Se han propuesto varios criterios de cobertura para generar los casos de prueba, entre ellos Cobertura de caminos (Path Coverage), el cual requiere que todos los caminos en el modelo de la aplicación web sean atravesados por algún caso de prueba; Cobertura de ramas (Branch Coverage), el cual requiere que todas las ramas en el modelo sean atravesadas por algún caso de prueba y Cobertura de nodos (Node Coverage), el cual requiere que todos los nodos en el modelo sean atravesados por algún caso de prueba.

Respecto a estas técnicas de caja blanca y al nivel en el que pueden aplicarse, la técnica propuesta por Liu et al es aplicable en varios niveles de prueba, desde pruebas unitarias hasta pruebas de integración. La técnica propuesta por Ricca y Tonella es aplicable exclusivamente en el nivel de pruebas de sistema.

Caja negra

Existen dos enfoques que aplican la estrategia de pruebas de caja negra para aplicaciones Web:

La primera fue propuesta por Di Lucca et al. [Di Lucca 2002] y consta de dos fases: la primera considera las pruebas unitarias de ítems dentro del modelo y la segunda fase considera pruebas de integración de ítems relacionados. Para la primera fase, el enfoque propone que se construyan tablas de decisión (Figura 4) para los ítems a ser probados (páginas Web, ya sean páginas clientes o de servidor). Estas tablas de decisión tienen dos secciones: sección de entrada y la sección de salida. La primera contiene condiciones en términos de variables de entrada para la página, acciones de entrada y el estado antes de la prueba, el cual está dado por valores globales de la aplicación, cookies y otros objetos Web. La sección de salida contiene los resultados

esperados, acciones de salida esperadas y el estado de la aplicación después de la prueba. La segunda fase integra varios ítems relacionados y se les aplica las mismas tablas de decisión. Los ítems que se agrupan son los que intervienen mayormente en un caso de uso.

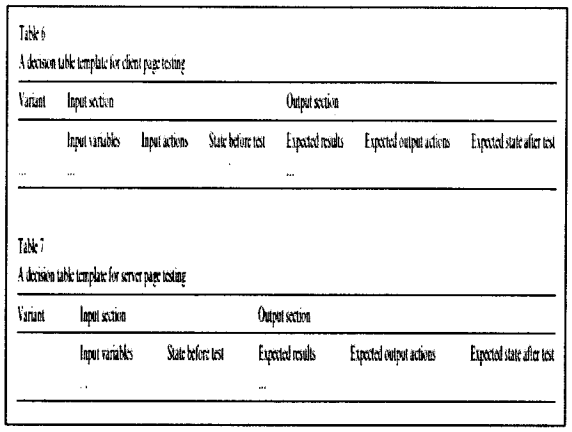

Table 6
A decision table template for client page testing

Variant	Input section			Output section		
	Input variables	Input actions	State before test	Expected results	Expected output actions	Expected state after test
...	...			...		

Table 7
A decision table template for server page testing

Variant	Input section		Output section		
	Input variables	State before test	Expected results	Expected output actions	Expected state after test
	...		...		

Fig. 4. Tablas de decisión [Di Lucca 2006]

La segunda técnica fue propuesta por Andrews et al. [Andrews 2005] y considera la aplicación de máquinas de estados finitos para modelar el comportamiento del software y generar casos de prueba a partir de ellos. Comprende dos fases: en la primera, la aplicación Web es modelada mediante una colección jerárquica de máquinas de estados finitos, donde el nivel más bajo está formado por las páginas Web y el nivel más alto representa la aplicación entera. En la segunda fase, los casos de prueba son generados a partir de esta representación.

Caja gris

Las estrategias de caja gris comprenden aquellas que están basadas en la recolección de información de sesión de usuario (user session data) o en la técnica de capturar y reproducir las acciones que un usuario realiza sobre la aplicación. Estas pruebas se usan principalmente cuando la aplicación evoluciona rápidamente y es necesario la ejecución de pruebas para validar que otras funcionalidades del software no hayan sido afectadas como consecuencia del mantenimiento o cambios realizados (pruebas de regresión). En esta estrategia se encuentra el enfoque basado en la generación de casos de prueba basados en sesiones de usuario.

4.4 Generación De Casos De Prueba Basadas En Sesiones De Usuario

En este enfoque, los casos de prueba son construidos en base a los archivos de bitácora del servidor Web [Sprenkle 2006]. Los archivos de bitácora (archivos log) están formados por todas las peticiones que ha atendido el servidor web, indicando la dirección IP del solicitante, el recurso solicitado y los parámetros usados en la llamada, como lo muestra la figura 5:

```
428.4.133.132 [17/Feb/2004:13:25:46 -0500]
GET /apps/bookstore/Login.jsp?
querystring=kret_page=%2Fapps%
2Fbookstore%2FShoppingCart.jsp ]
```

Figura 5: Ejemplo del contenido de un archivo de bitácora

A partir de esta información de construye las "sesiones de usuario", las cuales son secuencias de llamadas ejecutadas por un mismo usuario, de tal forma que permita simular el uso de la aplicación por parte de un usuario. Principalmente se usan cuando una aplicación ya se encuentra inicialmente desplegada, y se están haciendo los ajustes o modificaciones y se requiere que estas modificaciones no afectan a la funcionalidad ya construida (Pruebas de Regresión).

Tomando como base el framework propuesto por [Sprenkle 2007], se definen los siguientes mayores pasos para este mecanismo:

- Captura de la información de sesión de usuario
- Generación de los casos de prueba
- Reducción de casos de prueba
- Ejecución de los casos de prueba
- Evaluación de los resultados de la ejecución de los casos de prueba

A seguir describimos cada uno de estos pasos:

- **Captura de la información de sesión de usuario**

La principal ventaja de la técnica de pruebas basándose en sesión de usuarios es que la

información de los archivos de bitácora del servidor de aplicaciones puede ser fácilmente obtenida. No necesita realizarse ningún cambio en el código de la aplicación, sólo se modifica la configuración del servidor de aplicaciones. Por ejemplo, con el servidor de aplicaciones Resin se puede guardar la dirección IP del solicitante, el tipo de petición (GET/POST) y el recurso solicitado (nombre de la página web solicitada), así como también información adicional enviada (como argumentos, sus valores y cookies).

- **Generación de los casos de prueba**

El proceso de generar los casos de prueba se puede ver en la Figura 6. A partir de la información capturada en los archivos de bitácora, se construyen las sesiones de usuario, agrupando las solicitudes pertenecientes a una misma IP que estén dentro del tiempo de duración de la sesión de usuario configurada en el servidor de aplicaciones.

En el ejemplo mostrado en la figura, se nota que el archivo de bitácora contiene la fecha y hora de la solicitud, el IP del solicitante, y el recurso solicitado (en el ejemplo de la figura, los recursos que empiezan con "s" son estáticos, con "d" son recursos dinámicos y los que empiezan con "i" son irrelevantes).

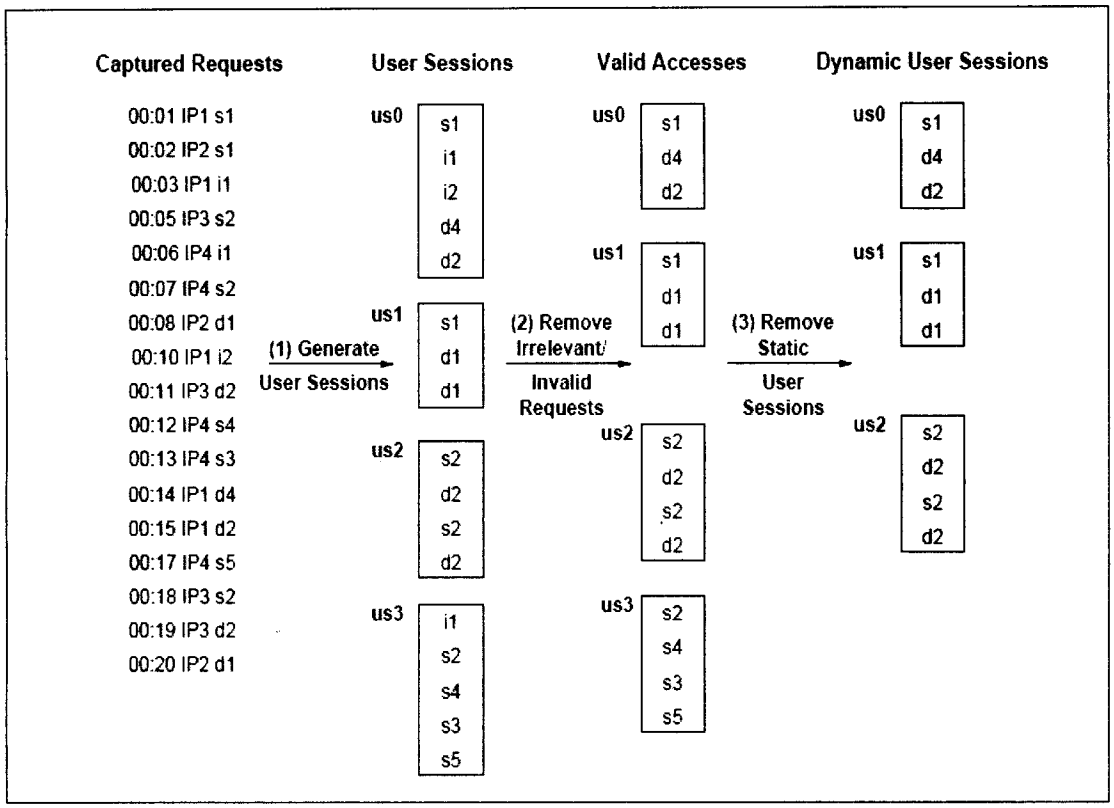

Figura 6: Ejemplo de generar casos de prueba a partir de información de sesiones de usuario [Sprenkle 2007]

- **Reducción de casos de prueba**

Para reducir el costo de mantener y ejecutar los casos de prueba obtenidos, antes de la ejecución esta suite de casos de prueba deben ser reducidos, considerando que la suite resultante debe tener las características de encontrar la mayor cantidad de errores como la tendría la suite original.
La figura 7 muestra el proceso de reducción de casos de prueba utilizando dos técnicas

disponibles para lograr el objetivo: La primera es la técnica de reducción basada en requerimientos y la segunda está basada en la utilización del análisis conceptual.

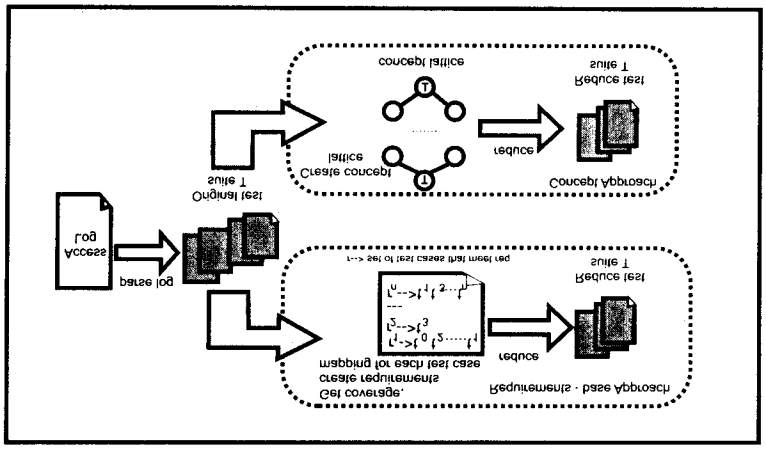

Figura 7: Proceso de reducción de casos de prueba [Sprenkle 2005]

- **Ejecución de los casos de prueba**
  Los casos de prueba generados son ejecutados por alguna herramienta de replay. Entre estas herramientas se encuentra la herramienta GNU wget [Wget 2008]. Para cada una de las solicitudes dentro del caso de prueba, wget envía la solicitud HTTP al servidor de aplicaciones web, incluyendo las variables de formularios y los valores de cada una de ellas.

- **Evaluación de los resultados de la ejecución de los casos de prueba**

  Para realizar la evaluación del resultado de la ejecución de los casos de prueba, se utiliza un oráculo (oracle), el cuál es una pieza de software que compara la salida del caso de prueba y la compara con la salida esperada (figura 8).

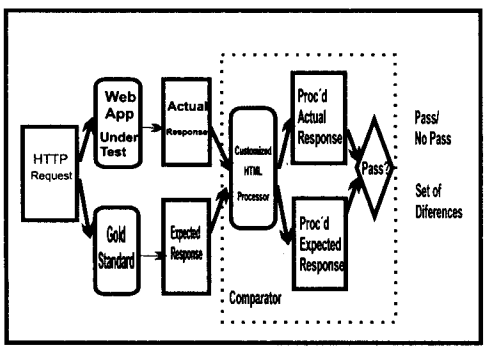

Figura 8: Procesos de un oracle [Sprenkle 2007]

5. REDUCCIÓN DE CASOS DE PRUEBA

Como se mencionó en la sección anterior, es necesario reducir el costo de mantener y ejecutar los casos de prueba obtenidos antes de su ejecución. El problema de reducir los casos de prueba puede ser establecido de la siguiente manera [Rothermel 2002]:

Dado: Una test suite T, un conjunto de requerimientos de casos de prueba r_1, r_2,...,r_n que deben ser satisfechos para proveer la cobertura deseada de pruebas al programa, y subconjuntos de T (T_1, T_2,...T_n), uno asociado con cada de los r_is de tal forma que cualquiera de los casos de prueba t_k perteneciente a T_i puede ser usado para probar r_i.

Problema: Encontrar un conjunto representativo de casos de prueba de T que satisfagan todos los ris.

5.1 Reducción de suites de prueba basada en requerimientos

El proceso de reducción de casos de prueba basado en requerimientos se plantea en la figura 9. Se puede notar en la figura, que a la suite original de casos de prueba se le aplica el proceso de reducción seleccionando casos de prueba para formar una suite de pruebas reducida. Esta suite de pruebas debe cumplir los requerimientos impuestos.

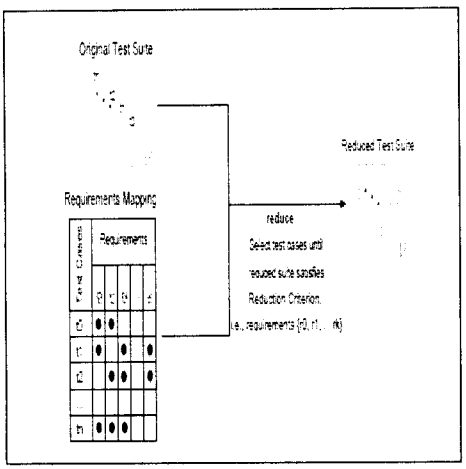

Figura 9: Proceso general de reducción de casos de prueba basado en requerimientos [Sprenkle 2005]

5.2 Criterio de reducción de suites de prueba y requerimientos

Un criterio de reducción de suites de prueba impone requerimientos en una suite reducida. Por ejemplo, un criterio de reducción comúnmente usado es "cobertura de sentencia". Una suite reducida satisface el criterio de "cobertura de sentencia" si por cada sentencia "s" que la suite de prueba original T cubre o ejecuta, al menos un caso de prueba "t" en la suite reducida T' ejecuta "s". En este ejemplo, las sentencias ejecutadas son los "requerimientos" que la suite reducida debe satisfacer para cumplir con el criterio "cobertura de sentencia".

Muchos criterios han sido propuestos, pero para las pruebas de aplicaciones Web se tomará el propuesto por Sampath et al [Sampath 2006] (criterio basado en URL), modificado por Sprenkle [Sprenkle 2007], cuyas variantes se muestran en la figura 10:

Test Case 1:

GET /bookstore/index.jsp

GET /bookstore/search.jsp?category=science&showresults=5

Test Case 2:

GET /bookstore/index.jsp

POST /bookstore/shoppingcart.jsp?item_no=XRVB588&price=50

POST /bookstore/index.jsp?cart=inuse

(a) Example Test Suite

Criteria	Form	Example Requirements for Figure 8.2(a)
resource (R)	resource	{/bookstore/index.jsp, /bookstore/search.jsp, /bookstore/shoppingcart.jsp}
resource+type (RT)	type resource	{GET /bpokstore/index.jsp, POST /bookstore/index.jsp, GET /bookstore/search.jsp, POST /bookstore/shoppingcart.jsp}
resource+names (RN)	resource parameter_names	{/bookstore/index.jsp, /bookstore/index.jsp?cart, /bookstore/search.jsp?category&showresults, /bookstore/shoppingcart.jsp?item_no&price}
resource+type+ names (RTN)	type resource parameter_names	{GET /bookstore/index.jsp, POST /bookstore/index.jsp?cart, GET /bookstore/search.jsp?category&showresults, POST /bookstore/shoppingcart.jsp?item_no&price}
resource sets (RSet)	sets of resources	{ (/bookstore/index.jsp, /bookstore/search.jsp), (/bookstore/index.jsp, /bookstore/shoppingcart.jsp) }

(b) URL-based Test Criteria

Figura 10: ejemplos de criterios de prueba basados en URL

La figura 10(a) muestra como ejemplo los dos casos de prueba que conforman nuestra suite de prueba, y la figura 10(b) los criterios de cobertura basados en URL propuestos por

Sampath et al. A continuación se describen cada uno de ellos:

- ✓ Resource (R): El criterio resource establece que todos los resources o recursos (objetos invocados) deben figurar en la suite de pruebas reducida.
- ✓ Resource+type (RT): El criterio resource+type establece que todos los resources o recursos (objetos invocados), considerando que un recurso puede ser invocado de manera diferente (GET o POST), deben figurar en la suite de pruebas reducida.
- ✓ Resource + names (RN): El criterio resource+names establece que todos los resources o recursos (objetos invocados), considerando que un recurso puede ser invocado con distintos nombres de parámetros, deben figurar en la suite de pruebas reducida.
- ✓ Resource+type+names (RTN): El criterio resource+type+names establece que todos los resources o recursos (objetos invocados), considerando que un recurso puede ser invocado de manera diferente (GET o POST) y con distintos nombres de parámetros, deben figurar en la suite de pruebas reducida.
- ✓ Resource sets (RSet): El criterio resource sets establece que cada conjunto de resources de cada caso de prueba debe figurar en la suite reducida.

La figura 11 muestra como ejemplo, los casos de prueba us1 hasta us6, cada uno de los cuales ejecuta o cubre los URL denotados como requerimientos. Como solo se denota los URL y no sus tipos o parámetros, se ajusta al criterio de cobertura de recursos. La figura 11 será usada para describir tres técnicas de reducción de casos de prueba en la sección siguiente.

Test Cases	Requirements						
	Def	Reg	Login	Logout	Shop	Books	MyInfo
us1	●	●	●				
us2	●	●	●		●		●
us3	●	●	●	●	●		
us4	●	●	●		●	●	
us5	●	●	●				
us6	●	●	●	●	●	●	

Figura 11: Ejemplo de mapeo entre casos de prueba y requerimientos para el criterio de cobertura de recursos

Enfoque Greedy

Este enfoque escoge el caso de prueba que provee la "mejora más marginal" de la actual suite de pruebas reducida en términos del criterio de selección. Es decir, selecciona casos de prueba de tal forma que los subsiguientes casos de prueba proveen máxima cobertura al criterio especificado. Cuando más de un caso de prueba tiene el mismo grado de "mejora más marginal", Greedy selecciona aleatoriamente uno de ellos.

Como ejemplo, usaremos este enfoque en el ejemplo de la Figura 11 para satisfacer el criterio de cobertura de recursos. Primero Greedy selecciona us6 para la suite reducida ya que us6 es el candidato que solicita la mayor cantidad de recursos (satisface la mayor cantidad de requerimientos). Después, ya que us2 provee la "mejora más marginal" respecto

al criterio de cobertura de recursos, us2 es añadida a la suite reducida. La combinación de us6 y us2 satisface el criterio de cobertura de recursos, por lo que Greedy reduce la suite a {us6, us2}.

Enfoque de Harrold, Gupta y Soffa (HGS)

El enfoque HGS usa una heurística para seleccionar casos de prueba representativos de la suite original. Está basada en la cardinalidad de requerimientos. El número de casos de prueba que cubren o ejecutan un requerimiento es la "cardinalidad del requerimiento". Después que el caso de prueba es añadido a la suite reducida, el algoritmo marca los requerimientos cubiertos por el caso de prueba. Entonces, el algoritmo selecciona el caso de prueba más ocurrido frecuentemente entre los casos de prueba no marcados con la más baja cardinalidad de requerimientos.

En caso de empates, el algoritmo escoge el caso de prueba que ocurre más frecuentemente en la próxima cardinalidad de requerimientos más alta.

En la Figura 11, el número de puntos en una columna representa la cardinalidad del requerimiento de recurso. HGS escogerá us2 primero, porque es la única sesión de usuario que cubre MyInfo (MyInfo tiene cardinalidad 1) y marca los requerimientos de MyInfo como satisfechos. El algoritmo entonces considera requerimientos con cardinalidad 2 (Logout y Books) y entonces selecciona el caso de prueba que ocurre más frecuentemente en la unión de estas dos columnas. Ya que us6 ocurre dos veces, mientras us3 y us4 ocurren sólo una vez, us6 es añadida a la suite reducida, y el algoritmo marca los requerimientos que us6 cumple. Y ya que el conjunto {us2, us6} cumple todos los requerimientos, HGS termina.

Enfoque de Análisis Conceptual

Análisis Conceptual es una técnica matemática para realizar clustering de objetos que tienen atributos discretos comunes. Sampath et al propusieron aplicar Análisis Conceptual en la reducción de casos de prueba para aplicaciones web. Si se toma como ejemplo la Figura 11, los objetos son los casos de prueba y los atributos son los recursos de la aplicación. Su funcionamiento se muestra en la Figura 12.

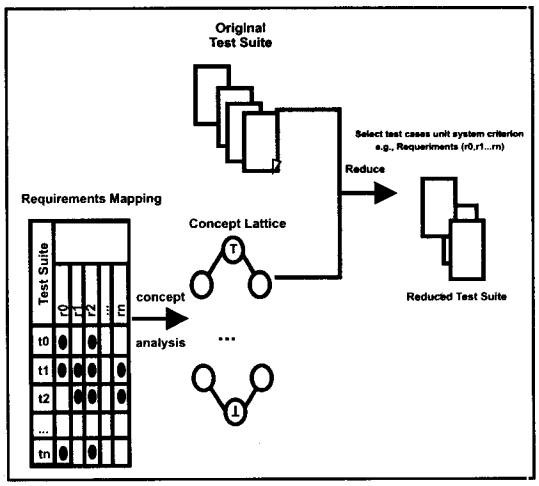

Figura 12: Proceso de reducción usando Análisis Conceptual

Como se muestra en la Figura 12, la reducción a través del Análisis Conceptual realiza un análisis para construir un "grafo de conceptos", donde cada nodo del grafo (o concepto) es una tupla (Oi,Ai), tal que todos los objetos en Oi (incluido en O) comparte todo y solo los atributos en Ai (incluido en A) y viceversa. Las esquinas del grafo denotan el orden parcial entre los nodos conceptos. Los conceptos que se encuentran en la parte inferior del grafo contienen objetos que son más similares (tienen más atributos en común) que los conceptos en la parte superior del grafo. La Figura 13 muestra en más detalle la representación del grafo de conceptos de la Figura 11.

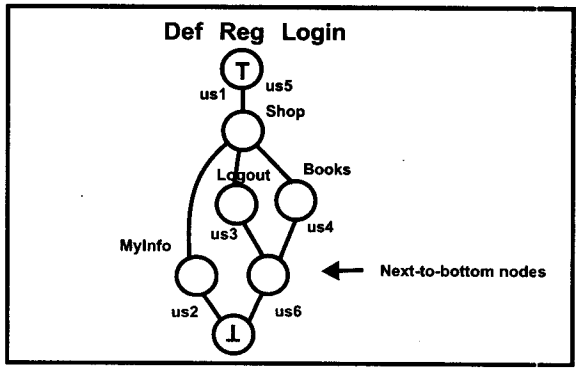

Figura 13: Grafo de Conceptos basado en la Figura 11

La heurística de Sampath selecciona aleatoriamente un caso de prueba del nodo inferior, un caso de prueba por cada concepto o nodo que está en el nivel superior del nodo

inferior (llamados nodos next-to-bottom). En la figura 13, los nodos next-to-bottom están etiquetados por us2 y us6. Aplicando la heurística, la suite reducida que satisface el criterio de cobertura de recursos es {us2,us6}.

6. CONCLUSIONES

- El presente trabajo hace una revisión de los métodos existentes para pruebas de aplicaciones web, haciendo un recorrido por los distintos tipos de pruebas (funcionales y no funcionales). El artículo profundiza en las pruebas del primer tipo. De las pruebas funcionales, se detallan las pruebas basadas en la generación de casos de prueba basadas en "sesiones de usuario". Este tipo de prueba puede ser automatizada evitando la participación humana en la generación de los casos de prueba o la evaluación de los resultados.

- Para la realizaci de estas pruebas, es necesario antes, reducir los casos de prueba generados debido a que hay información extensa y redundante. Se han presentado en el presente trabajo, tres técnicas para realizar esta reducción. Estas técnicas de reducción deben garantizar que los casos de prueba reducidos tengan la misma capacidad de encontrar errores en el software que la suite original.

REFERENCIAS BIBLIOGRÁFICAS

[Andrews 2005] Anneliese A. Andrews, Jeff Offutt, Roger T. Alexander. "Testing Web Applications by Modeling with FSMs". *Software and Systems Modeling,* vol. 4, no. 3, pp. 326-345, July 2005.

[Di Lucca 2006] Giuseppe A. Di Lucca, Anna Rita Fasolino. "Testing Web-based applications: The state of the art and future trends". *Information & Software Technology*, Vol. 48, Number 12, pp.1172-1186, December 2006.

[Elbaum 2003] Sebastian Elbaum, Srikanth Karre, Gregg Rothermel. "Improving Web Application Testing with User Session Data". *Proceedings of 25th International Conference on Software Engineering.* pp. 49-59, May 2003.

[Hao 2006] Jianhua Hao, Emilia Mendes. "Usage-based Statistical Testing of Web Applications". *Proceedings of the 6th international conference on Web engineering. ACM International Conference Proceeding Series.* Vol. 263. pp. 17-24, July 2006.

[Liu 2000] C. Liu, D.C. Kung, P. Hsia, C. Hsu. "Object-based data flow testing of Web Applications". *Proceeding of First Asia Pacific Conference on Quality Software*, pp 7-17, Hong Kong, China, 2000.

[Offutt 2002] Jeff Offutt. "Quality attributes of Web software applications". *Software, IEEE*, 19(2), pp. 25-32, April 2002.

[Ricca 2001] F. Ricca y P.Tonella. "Analisys and testing of Web applications". *Proceeding of 23rd International Conference on Software Engeneering*, pp. 25-34, 2001.

[Rothermel 2002] G. Rothermel, M.J. Harrold, J. von Ronne, and C. Hong. "Empirical Studies of Test-Suite Reduction," *Software Testing Verification and Reliability*, Vol. 12, No. 4, 2002, pp. 219-249.

[Sampath 2006] Sreedevi Sampath, Sara Sprenkle, Emily Gibson, Lori Pollock. "Web application testing with customized test requirements – an experimental comparison study". *Proceeding in 17th International Symposium on Software Reliability Engineering* (ISSRE 2006), pp 266-278, November 2006.

[Sant,2005] Jessica Sant; Amie Souter; Lloyd Greenwald. "An Exploration of Statistical Models for Automated Test Case Generation". *Proceeding of the Third International Workshop on Dynamic Analysis* (WODA 2005) May 2005. pp. 1-7.

[Sprenkle 2006] Sara Sprenkle, Emily Gibson, Sreedevi Sampath, and Lori Pollock. "A Case Study of Automatically Creating Test Suites from Web Application Field Data". *Procedding of Workshop on Testing, Analysis and Verification of Web Software* 2006, pp. 1-9.

[Sprenkle 2005] Sara Sprenkle, Sreedevi Sampath, Emily Gibson, Lori Pollock, Amie Souter. "An Empirical Comparison of Test Suite Reduction Techniques for User-session-based Testing of Web Applications". *Proceeding of*

21st IEEE International Conference on Software Maintenance 2005. pp. 587 - 596.

[Sprenkle 2007] Sara Sprenkle, "Strategies for automatically exposing faults in Web applications". *PhD Thesis in Computing and Information Science*, University of Delaware, Newark, 2007.

[Wget,2008] GNU Wget. http://www.gnu.org/software/wget/wget.htm, revisado en 02-02-2009.