UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Generación de escenarios usando *WebSpec*
Autores: Cristian Eric Cianfagna
Director: Gustavo Rossi
Codirector: -
Asesor profesional: -
Carrera: Licenciatura en Sistemas

Resumen

El desarrollo de aplicaciones WEB es un proceso complejo que consume tiempo e involucra diferentes roles. Dada su naturaleza, este tipo de aplicaciones tienen características únicas, tales como acceso a la información por medio de la navegación y características sofisticadas de interacción. Sin embargo, pocas herramientas se han propuesto para representar este tipo de requerimientos que son específicos de las aplicaciones WEB. *WebSpec* es un lenguaje de dominio específico cuya función es especificar las características más relevantes de los requerimientos de aplicaciones WEB: interacción y navegación. La mayoría de las herramientas de software que se utilizan en la ingeniería de requerimientos WEB requieren un cierto grado de formalidad para su utilización. Esta formalidad necesaria es una de las principales barreras con la que se encuentra el usuario a la hora de utilizarlas. Los diagramas de *WebSpec* son el principal artefacto para representar escenarios de uso. Estos requieren cierta formalidad para su correcta construcción. Es por esta formalidad que el usuario que utiliza por primera vez *WebSpec* se encuentra con algunas dudas. Para abordar la problemática de entendimiento de la utilización de *WebSpec*, se propone la generación de documentos que facilitan la comprensión de los requerimientos que se van especificando en *WebSpec*.

Palabras Claves

WebSpec, requerimientos web, interacción, navegación, transformación de modelos

Trabajos Realizados

Se realizó un estudio sobre el estado del arte de las herramientas que ofrecen la posibilidad de generar documentos de escenarios de uso.
Se implementó en *WebSpec* un módulo para generar documentos de escenarios de uso dado un diagrama de *WebSpec*.
Se realizó un caso de estudio por el cual se utilizaron la mayoría de las características de WebSpec (principalmente el nuevo módulo de generación de escenarios).

Conclusiones

WebSpec posee una característica más, cuyo objetivo es que el usuario cuente con una alternativa para poder ir comprendiendo los diagramas que va realizando y pueda también obtener un documento que pueda ser utilizado como documentación y medio de comunicación. Los documentos con formato tabular (HTML y WORD) son apropiados como comprensión y comunicación de los diagramas, y el formato enumerativo presenta una mejor estructura desde el punto de vista de la documentación.

Trabajos Futuros

Entre los trabajos futuros se encuentran:

- Mejorar el manejo de imágenes de los documentos de tipo *WORD.*
- Mejorar la estructura tabular de los documentos de tipo *WORD.*
- Ofrecer un nuevo tipo de documento, por ejemplo ppt.
- Ofrecer la posibilidad de generar documentos de tipo PDF.
- Soporte para nuevos idiomas.

Fecha de la presentación: JUNIO 2013

Facultad de Informática
Universidad Nacional de La Plata

Generación de escenarios usando *WebSpec*

Tesina para obtener el grado de
Licenciatura en Sistemas

Autor: Cristian Eric Cianfagna
Director: Gustavo Rossi

La Plata - Bs. As.
Argentina
Junio 2013

A mi madre y a mi abuela, sin ellas esto nunca hubiese sido posible.

AGRADECIMIENTOS

Quiero agradecer al Dr. Gustavo Rossi, por permitir desarrollar mi tesis bajo su tutoría y especialmente al Dr. Esteban Robles Luna por sus aportes durante el desarrollo del trabajo y la escritura de este documento.

ABSTRACT

El desarrollo de aplicaciones WEB (aplicaciones que corren sobre navegadores) es un proceso complejo que consume tiempo e involucra diferentes roles (desde clientes hasta desarrolladores). Dada su naturaleza, este tipo de aplicaciones tienen características únicas, tales como acceso a la información por medio de la navegación y características sofisticadas de interacción. Sin embargo, pocas herramientas se han propuesto para representar este tipo de requerimientos que son específicos de las aplicaciones WEB.

WebSpec es un lenguaje de dominio específico cuya función es especificar las características más relevantes de los requerimientos de aplicaciones WEB: interacción y navegación.

La mayoría de las herramientas de software que se utilizan en la ingeniería de requerimientos WEB requieren un cierto grado de formalidad para su utilización. Esta formalidad necesaria es una de las principales barreras con la que se encuentra el usuario a la hora de utilizarlas.

Los diagramas de *WebSpec* son el principal artefacto para representar escenarios de uso. Estos requieren cierta formalidad para su correcta construcción. Es por esta formalidad que el usuario que utiliza por primera vez *WebSpec* se encuentra con algunas dudas.

Para abordar la problemática de entendimiento de la utilización de *WebSpec*, se propone la generación de documentos que facilitan la comprensión de los requerimientos que se van especificando en *WebSpec*.

En este trabajo, se presenta el desarrollo de una herramienta de software que genera una serie de documentos con distintos formatos y estructuras. Estos contienen todos los escenarios de uso que se van especificando con *WebSpec* y están expresados de tal forma que el usuario pueda comprenderlos.

ÍNDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN

El desarrollo de aplicaciones WEB (aplicaciones que corren sobre navegadores) es un proceso complejo que consume tiempo e involucra diferentes roles (desde clientes hasta desarrolladores). Dada su naturaleza, este tipo de aplicaciones tienen características únicas, tales como acceso a la información por medio de la navegación y características sofisticadas de interacción. Sin embargo, pocas herramientas se han propuesto para representar este tipo de requerimientos que son específicos de las aplicaciones WEB.

Este primer capítulo tiene por objetivo exponer el contexto por el cual se motivó la presente tesis. Para ello, en primer lugar se introducen procesos, herramientas y técnicas que se utilizan para el desarrollo de software. Luego, se presenta *WebSpec* (una herramienta para tratar requerimientos) y su campo de acción dentro del proceso de desarrollo de software. Posteriormente, se observan algunas falencias de la utilización de *WebSpec* en el contexto inicialmente presentado; y finalmente se realiza una propuesta para abordar las falencias observadas.

La organización del presente capítulo se conforma de la siguiente manera: en la Sección 1.1, se habla de metodologías ágiles y de cómo están focalizadas en acercar a los usuarios con el equipo de desarrollo. El proceso iterativo y evolutivo que siguen estas es presentado, para ello introducimos el concepto de iteración o *sprint*. Posteriormente, se introducen las historias de usuario como una de las principales herramientas para representar

requerimientos. En la Sección 1.2, se presenta *WebSpec*. A continuación, en las secciones finales, se presenta un contexto para ejemplificar la problemática (Sección 1.3), una serie de observaciones y la motivación de esta tesis (Sección 1.4), la propuesta (Sección 1.5) para abordar la problemática planteada y la estructura de la presente tesis (Sección 1.6).

1.1 BACKGROUND

En esta sección, se definen metodologías ágiles, iteraciones o *sprints* e historias de usuario, y además se establece de qué manera se relacionan. El objetivo principal es situar estas herramientas en el proceso de desarrollo de software.

1.1.1 METODOLOGÍAS ÁGILES

Las metodologías ágiles son una alternativa a la gestión tradicional de proyectos de desarrollo de software (desarrollo en cascada o secuencial).

Durante los años ochenta, la mayor parte del desarrollo de software era una actividad desorganizada. A menudo era una actividad caracterizada por las acciones “escribe código y corrige código”. El software era escrito sin mucha planificación, y su diseño se realizaba mediante la toma de decisiones rápidas. Esta tendencia para el desarrollo de software funcionaba bien cuando el software por construir era pequeño y sin demasiada complejidad. Pero los problemas aparecían cuando el software comenzaba a crecer, dado el agregado de nueva funcionalidad. Más aún, empezaban a aparecer errores y la dificultad aparejada debido a la corrección de éstos. Una clásica señal de este tipo de situación era pasar por una etapa extensiva de validación antes de su instalación

en el ambiente productivo. Este tipo de etapas extensivas impactaban directamente en los cronogramas de planificación.

Tiempo después, surgió la noción de “metodologías” en el desarrollo de software. Éstas le imponían un proceso matemático y estructurado con el objetivo de que el desarrollo de software fuese una actividad predecible y eficiente. Seguían un proceso detallado con un fuerte énfasis en la planificación, inspirado por otras disciplinas de la Ingeniería. Éstas se solían llamar “metodologías dirigidas por la planificación”, pero no han tenido una gran popularidad. La mayor crítica que se les hacía se debía a la burocracia que demandaban. Había que realizar mucho trabajo para poder aplicarlas, y, por ende, el desarrollo de software propiamente dicho consumía más tiempo del esperado.

Las metodologías ágiles surgen como una reacción a estas metodologías. Se atribuyen este surgimiento debido a la extremada burocracia de las metodologías de las ingenierías. Las metodologías ágiles intentan establecer un compromiso entre casi no tener procesos y tener demasiados procesos, y simplemente ofrecen el proceso mínimo necesario para lograr un resultado razonable. Como consecuencia, presentan cambios significativos con respecto a los métodos de las ingenierías. Una de las diferencias notables es que tienen una menor documentación. En muchos sentidos, son más orientadas a código, siguen la premisa que dice que la parte fundamental de la documentación es el código fuente. La falta de documentación es una señal de otras dos diferencias:

- Los métodos ágiles son más adaptables que predecibles. Los métodos de las ingenierías tienden a planificar el proceso de desarrollo en detalle. Esto funciona hasta que los cambios empiezan a surgir. Sin embargo, los métodos ágiles reciben los cambios con los brazos abiertos. Tratan de ser procesos que se adaptan y prosperan hasta el punto de cambiarse a sí mismos.
- Los métodos ágiles están más orientados a personas que a procesos. El objetivo de los métodos de las ingenierías consiste en definir un proceso

que va a funcionar bien cualquiera sea la persona que lo use, mientras que el objetivo de los métodos ágiles es presentar un proceso para brindar apoyo al equipo sin modificar su perfil.

1.1.2 ITERACIÓN O *SPRINT*

Actualmente, las iteraciones de desarrollo de software están dirigidas por metodologías ágiles; éstas proponen una nueva versión de la aplicación en el ambiente productivo cada tres y cinco semanas. A lo largo del presente documento, no se hará diferencia a la hora de hacer referencia a las palabras iteración o *sprint*. Por una cuestión de consistencia, se opta por la palabra iteración.

En el desarrollo de software, una iteración es un período durante el cual determinada cantidad de trabajo debe ser completada y preparada para una revisión. Cada iteración comienza con una reunión de planificación. Durante la reunión, el responsable del producto (la persona que requiere el trabajo) y el equipo de desarrollo acuerdan qué trabajo será completado durante la iteración. El equipo de desarrollo tiene la última palabra con respecto a la cantidad de trabajo que puede afrontar durante la iteración, y el responsable del producto tiene la última palabra con respecto a qué criterios hay que cumplir para que el trabajo sea aprobado y aceptado.

Una vez que el equipo dio consenso de cuántos días debe durar una iteración, se especifica el mismo período para todas las iteraciones. Una vez que la iteración comienza, el responsable del producto toma un rol pasivo y deja al equipo realizar el trabajo. Durante la iteración, el equipo celebra reuniones diarias de no más de 15 minutos para exponer el estado de las tareas de cada uno de los integrantes del equipo. En estas reuniones, el responsable del producto puede estar presente como observador. En el final de la iteración, el equipo presenta el trabajo completo al responsable del producto, y éste valida aquél con los criterios establecidos.

1.1.3 HISTORIAS DE USUARIO

Una historia de usuario es una representación de un requerimiento de software escrito en una o dos frases con el lenguaje común del usuario. En la Sección 1.1.2, se definió que una iteración es un período en el cual una determinada cantidad de trabajo debe ser realizada. Esta determinada cantidad de trabajo (requerimientos de software) está representada por medio de historias de usuario.

Las historias de usuario son una de las herramientas más populares para especificar requerimientos en las metodologías ágiles. En ejemplos tales como *SCRUM* [SBM01] y *XP* [BXP00], las historias de usuario son una herramienta esencial de desarrollo.

Las historias de usuario comparten el mismo propósito que los ya conocidos casos de uso [UCIJ02]; sin embargo, no son similares. Éstas son utilizadas para crear estimaciones de tiempo en las reuniones de planificación de nuevas versiones, también para documentar requerimientos en lugar de tener grandes volúmenes de documentación. Las historias de usuario son escritas por los mismos usuarios, que cuentan cuáles son las cosas que el sistema debe realizar en su favor. Se pueden ver como escenarios de uso, excepto que no están limitadas a describir solo la UI. Están escritas en un formato de texto informal de, a lo sumo, 3 sentencias sin ningún tipo de vocabulario técnico.

Las pruebas de aceptación son originadas debido a las historias de usuario. Una serie de pruebas de aceptación automatizada debe ser creada para validar que la implementación de una historia de usuario haya sido realizada correctamente.

Una de las mayores diferencias con respecto a las especificaciones de requerimiento tradicionales es el nivel de detalle. Las historias de usuario deberían proveer solamente el detalle necesario como para poder realizar una estimación de riesgo y de cuánto tiempo llevará la historia para poder ser

implementada. Cuando el momento de implementar la historia llega, el desarrollador se junta con el usuario para obtener una detallada descripción de los requerimientos. De esta manera, los desarrolladores estiman cuánto tiempo les va a llevar implementar las historias.

Otra diferencia entre las historias y los documentos de requerimientos es el hincapié sobre las necesidades del usuario. Se debería evitar la especificación de detalles técnicos, tecnológicos, bases de datos y algoritmos. También se debería evitar especificaciones de UI, y se debería focalizar en los beneficios y en las necesidades del usuario.

1.2 *WEBSPEC*

WebSpec [RLRG1] es un lenguaje de dominio concreto para la especificación de los aspectos más relevantes y característicos de los requerimientos de aplicaciones WEB (aplicaciones ejecutadas sobre navegadores); éstos involucran interacción y navegación.

Algunas de las características de *WebSpec* que se presentan con mayor detalle en el Capítulo 3 son: diagramas, especificación de interacción, navegación, simulación de escenarios y generación automática de tests y código de UI (Interfaz Gráfica de Usuario).

WebSpec encuentra su principal campo de acción en la ingeniería de requerimientos WEB. De la misma forma que con las historias de usuario y casos de uso, con *WebSpec* podemos obtener, especificar y validar requerimientos WEB. El diagrama de *WebSpec* es la principal herramienta por la cual se puede especificar escenarios de uso de las aplicaciones WEB.

A lo largo de los últimos años, han surgido varias metodologías para mejorar el proceso de desarrollo de aplicaciones WEB (aplicaciones ejecutadas sobre navegadores). Todas estas metodologías siguen un estilo *top-down* y son dirigidas por modelos [MDWE]. La mayoría define una secuencia de etapas, y en

cada una de estas se especifica un modelo propio. Dada la secuencia de las etapas, cada uno de estos modelos va sufriendo una serie de transformaciones a medida que va pasando por las distintas etapas.

Por otro lado, las metodologías ágiles están basadas en una fuerte relación e interacción con el cliente. La idea es hacer participar al cliente en cada una de las etapas que sea posible. Para lograrlo, tenemos que entenderlo y que nos entienda. Para ello, necesitamos un lenguaje un poco más formal, no simplemente texto o bocetos gráficos en papel.

La idea es promover un lenguaje formal como *WebSpec* por el cual el cliente pueda comunicarse de una manera mucho más concreta y pueda participar en distintas etapas e interactuar con los distintos roles que intervienen en el desarrollo de aplicaciones WEB.

1.3 CONTEXTO

En esta sección, se presenta un contexto que sirve de ejemplo para realizar algunas observaciones e identificar la problemática (Sección 1.4) que se aborda en esta tesis. El ejemplo que se describe es una iteración en el ciclo de desarrollo de un equipo.

Se toma una iteración de cuatro semanas, que se transforman en veinte días de trabajo. Esta iteración se podría componer de la siguiente forma: una etapa de relevamiento de requerimientos, otra de desarrollo de requerimientos (llamada implementación) y, por último, una etapa de validación de los requerimientos desarrollados (llamada aseguramiento de la calidad). La etapa de implementación tendrá una duración de diez días y, por consiguiente, tanto la etapa de relevamiento como la de aseguramiento de la calidad tendrán una duración de cinco días.

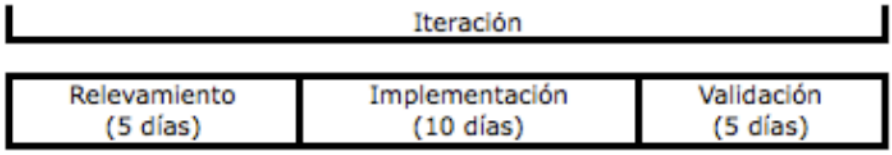


Fig. 1. Iteración

Por lo tanto, durante los primeros cinco días en los que la etapa de relevamiento tendrá lugar, el analista deberá: reunirse con el cliente, identificar los requerimientos, validar estos requerimientos identificados y, por último, transcribirlos en alguna herramienta para que los desarrolladores, posteriormente, puedan interpretar qué realizar. Alternativas de este tipo de herramientas son las historias de usuario, casos de usos y diagramas de *WebSpec* (en el Capítulo 3 se presentan en detalle).

La etapa de relevamiento comienza cuando el usuario escribe sus necesidades en papel. De estas necesidades surgen las especificaciones de requerimientos (como ya mencionamos, podrían ser representadas por historias de usuario, casos de uso o diagramas de *WebSpec*). Para que la tarea de identificación sea realizada de la mejor manera, el analista se reúne con un diseñador para armar algunas maquetas gráficas (llamadas *mockups*) por medio de las cuales el cliente valida el relevamiento de los requerimientos. No descartamos la posibilidad de que el analista pueda interactuar con un desarrollador para verificar la viabilidad de los nuevos requerimientos. Se debe tener presente que solamente se tienen cinco días para realizar todo lo anteriormente mencionado. La interacción entre los distintos roles en esta primer etapa también es crucial, dado que los lenguajes utilizados por cada uno de los roles involucrados no son iguales. Finalmente, el analista completa con mayor detalle cada una de las historias de usuario en base a todas las especificaciones recopiladas anteriormente.

Durante la etapa de implementación, los desarrolladores implementan cada uno de los requerimientos que fueron elegidos y estimados (éstos se decidieron en la reunión de planificación de la versión e iteración) para realizar durante los diez días de duración de esta etapa. Una vez que esta etapa se

culmina, se inicia la etapa de aseguramiento de la calidad en la cual se validan los requerimientos implementados.

En la etapa de aseguramiento de la calidad, el encargado de comprobar las validaciones utiliza como referencia la especificación de los requerimientos que el analista realizó en la primera etapa de la iteración. Una vez culminada esta etapa, se preparan todos los requerimientos para ser puestos en el ambiente productivo. Luego, el equipo se prepara para comenzar con una nueva iteración.

1.4 PROBLEMÁTICA

Se han analizado varios casos de uso (contextos similares al presentado en la Sección 1.3) de *WebSpec* con diferentes tipos de usuarios, desde usuarios con un amplio conocimiento en el desarrollo de software hasta usuarios con muy poca interacción con sistemas informáticos. Basados en toda esta experiencia de uso de *WebSpec,* se nota que al usuario promedio le cuesta bastante asimilar los diagramas, así como sus conceptos principales. Básicamente, el problema con los diagramas se comienza a manifestar en aquellos usuarios que utilizan por primera vez *WebSpec*. Las interacciones y navegaciones no son lo suficientemente expresivas en cuanto a su objetivo a la hora de especificar escenarios de uso por parte de usuarios principiantes. Esto hace que los usuarios duden y requieran de una persona o usuario con más experiencia en *WebSpec*. Otra situación notoria con los usuarios primerizos es que les cuesta identificar los distintos escenarios que están representados en un diagrama.

A continuación se formula una propuesta para abordar esta problemática.

1.5 PROPUESTA

Planteada la problemática, vamos a proceder a formular una propuesta para abordar las observaciones de la Sección 1.4.

La idea es proveer una herramienta que le permita al usuario validar los conceptos que está utilizando en los primeros intentos de construir diagramas con *WebSpec*. Para ello se propone una generación textual en base al diagrama que se está construyendo. Se proponen tres estrategias de derivación de diagramas en escenarios de uso: dos basadas en documentos de tipo WORD y una de tipo HTML.

Las estrategias de WORD difieren en la organización del contenido dado que una presenta la derivación de diagramas basada en un formato tabular (con tablas) y la otra en un formato enumerativo (con secciones y subsecciones). La estrategia de HTML es similar a la estrategia de WORD basada en tablas, la diferencia radica en el aprovechamiento de todas las características disponibles que podemos utilizar a la hora presentar y organizar contenido en un documento HTML.

De esta manera, el usuario obtiene descripción textual de los escenarios de uso del diagrama que va construyendo. La derivación textual es una herramienta que se agrega a *WebSpec* con el fin de brindarle al usuario una alternativa de compresión de los escenarios que puede ser impresa o exportada digitalmente.

1.6 ESTRUCTURA DE LA TESIS

La estructura del presente documento se describe brevemente a continuación.

En el próximo capítulo (Capítulo 2), se muestran algunos trabajos similares, en los cuales se plantea la necesidad de transformar un modelo en otro.

Una vez vistas varias alternativas de transformaciones, se presentan los conceptos fundamentales de *WebSpec* (Capítulo 3). En este capítulo, se brinda la posibilidad de familiarizarse con cada una de las herramientas con las que *WebSpec* cuenta para un efectivo manejo de requerimientos.

Posteriormente (Capítulo 4), se plantea la solución conceptual a la problemática presentada en la sección 1.4. Luego (Capítulo 5), se muestra la implementación realizada para la generación de escenarios.

Posteriormente, se presenta un caso de estudio (Capítulo 6), en el cual se muestran las características y herramientas de *WebSpec* durante una iteración y, por último, un capítulo de conclusiones y trabajo futuro (Capítulo 7).

CAPÍTULO 2

TRABAJO RELACIONADO

Este capítulo presenta el estado del arte de la ingeniería de requerimientos. En el desarrollo del capítulo, se hace una comparación entre distintas alternativas que se han analizado en el campo de la ingeniería WEB, además de las diferentes técnicas y modelos que utilizan para llevar adelante la ingeniería de requerimientos.

El capítulo comienza con una breve introducción (Sección 2.1), luego hacemos mención del proceso de ingeniería de requerimientos (Sección 2.2), así como de las principales técnicas para capturar, especificar y validar requerimientos. Posteriormente, algunas de las principales metodologías WEB son descriptas (Sección 2.3). Luego, las alternativas son clasificadas y comparadas (Sección 2.4). Finalmente, se presentan las conclusiones del capítulo (Sección 2.5).

2.1 INTRODUCCIÓN

Debido a la Internet, las aplicaciones WEB en la actualidad son masivamente consumidas y demandadas. Esta tendencia, en los últimos años, ha despertado y generado la necesidad de metodologías de desarrollo que provean un apropiado soporte para la construcción de aplicaciones WEB. En los últimos años, distintos grupos de investigación han propuesto metodologías con

procesos, modelos y técnicas para construir este tipo de aplicaciones [ET02, K99]. Sin embargo, la mayoría de todas estas metodologías están concentradas en la etapa de diseño de los ciclos de vida, mientras que otras tareas como ingeniería de requerimientos, tests y calidad son manejadas con menor o casi ningún interés.

En el desarrollo de aplicaciones tradicionales (no WEB), profesionales y expertos en procesos consideran a la ingeniería de requerimientos como la etapa más importante del proceso de desarrollo. Los errores comunes, los complicados de encontrar y los que son difíciles de corregir son usualmente errores consecuentes de una inadecuada ingeniería de requerimientos. Muchas técnicas han sido propuestas, algunas para capturar requerimientos tales como entrevistas, *storyboarding*, algunas para especificar requerimientos tales como escenarios, historias de usuario o casos de uso y hasta para validación de requerimientos relevados, como por ejemplo, prototipos.

Aunque la importancia, los beneficios y las ventajas de una buena ingeniería de requerimientos son conocidos, estas técnicas son aplicadas con muy poca eficacia en el campo de la ingeniería WEB. Por el contrario, las aplicaciones WEB necesitan un proceso mucho más extenso y detallado de ingeniería de requerimientos debido a la cantidad de participantes involucrados y debido a la distinta variedad de requerimientos (tales como requerimientos de navegación, procesos de negocio, usabilidad, etc.).

2.2 TÉCNICAS DE INGENIERÍA DE REQUERIMIENTOS

Un requerimiento se define como una condición o capacidad que el sistema debe cumplir para satisfacer un contrato, una especificación, o alguna otra documentación formal. Los requerimientos definidos para un sistema deben ser: correctos, consistentes, verificables y trazables. La ingeniería de requerimientos es un proceso de obtención, comprensión, especificación, y

validación de los requerimientos de clientes y usuarios. También define las restricciones tecnológicas bajo las cuales el sistema debe construirse y funcionar. Es un proceso iterativo y cooperativo con el propósito de analizar el problema, documentar los resultados de distintas formas y evaluar la precisión de los resultados obtenidos.

Independientemente del tipo de aplicación de software a ser construida, el equipo de desarrollo tiene que adquirir un cierto conocimiento sobre el dominio del problema y los requerimientos de la aplicación. La obtención y especificación de estos requerimientos es un proceso complejo, dado que se debe identificar la funcionalidad que el sistema tiene que cumplir para satisfacer las necesidades de los usuarios y clientes.

Aunque no existe un proceso estandarizado que dé soporte al proceso de manejo de requerimientos y garantía de calidad de resultados, existen buenas prácticas en el desarrollo de aplicaciones que proveen un conjunto de técnicas. Éstas son recomendadas por algunas metodologías WEB para la especificación de requerimientos de aplicaciones WEB. Es importante notar que la selección de estas técnicas es responsabilidad del equipo de desarrollo, y el éxito de los resultados depende de la participación conjunta de clientes y usuarios que participan en el proceso.

El proceso iterativo de ingeniería de requerimientos consiste en tres actividades principales:

1. Obtención de requerimientos
2. Especificación de requerimientos
3. Validación de requerimientos

El proceso comienza con la obtención de requerimientos. Un grupo de desarrolladores recolecta la información de los usuarios y clientes. Esta información puede ser obtenida de diferentes fuentes, tales como documentos, aplicaciones *legacy*, entrevistas, etc., que son utilizadas en la preparación de la lista de requerimientos. Finalmente, la validación de requerimientos es llevada a

cabo para tratar de descubrir inconsistencias, errores o requerimientos que no fueron definidos. El proceso de especificación y validación es iterativo, puede llevarse a cabo varias veces en proyectos complejos.

En las próximas secciones, se describen brevemente algunas técnicas clásicas para la obtención, especificación y validación de requerimientos. Estas técnicas pueden ser más o menos adecuadas para las ingenierías de requerimientos WEB. Es muy difícil establecer criterios precisos para seleccionar las técnicas adecuadas. Estos criterios pueden incluir la facilidad de aprendizaje y uso, el costo, la calidad de los resultados y hasta el tiempo requerido para aplicar estas técnicas. Por ejemplo, el uso del lenguaje natural en la especificación de requerimientos da resultados menos precisos que una especificación hecha con casos de uso, la cual es menos precisa si los requerimientos se especifican por medio de un lenguaje formal.

2.2.1 OBTENCIÓN DE REQUERIMIENTOS

La recolección de requerimientos es una actividad en la cual el equipo de desarrollo identifica, de cualquier fuente disponible, la funcionalidad que el sistema necesita proveer para los futuros usuarios. Esta actividad también suele ser conocida como captura de requerimientos, descubrimiento de requerimientos o adquisición de requerimientos. El proceso de obtención de requerimientos puede ser complejo, especialmente si el dominio del problema no es conocido por los analistas.

A continuación se presentan algunas de las más relevantes técnicas usadas para la obtención de requerimientos en el contexto de un proceso de desarrollo de software estándar:

- Entrevistas
- JAD (*Joint Application Development*)
- *Brainstorming*

- Mapeo de conceptos (*Concept Mapping)*
- *Storyboarding*
- Casos de uso
- Cuestionarios

La comunidad de ingeniería de requerimientos ha propuesto muchas otras alternativas, tales como el análisis de sistemas o documentación similar. Sin embargo, la lista anteriormente descripta proporciona un conjunto representativo de las técnicas frecuentemente utilizadas.

2.2.2 ESPECIFICACIÓN DE REQUERIMIENTOS

Para la actividad de especificación de requerimientos, muchas técnicas han sido propuestas. En esta sección, enumeramos las más utilizadas. Dado el alcance de esta tesis, solamente se ofrece una descripción de las técnicas.

- **Lenguaje natural (Historias de usuario):** Resulta una técnica muy ambigua para la definición de los requerimientos. Consiste en definir los requerimientos en lenguaje natural sin usar reglas para ello. A pesar de que son muchos los trabajos que critican su uso, es cierto que a nivel práctico se sigue utilizando.
- **Glosario y Ontologías:** La diversidad de personas que forman parte de un proyecto software hace que sea necesario establecer un marco de terminología común. Esta necesidad se presenta en los sistemas de información WEB, puesto que los equipos de desarrollo suelen ser más interdisciplinarios. Por esta razón, son muchas las propuestas que abogan por desarrollar un glosario de términos en el que se recogen y definen los conceptos más relevantes y críticos para el sistema. En esta línea se encuentra también el uso de ontologías, en las que no sólo aparecen los términos, sino también las relaciones entre ellos.

- **Patrones o Plantillas (*Templates*):** Esta técnica, recomendada por varios autores, tiene por objetivo describir los requerimientos mediante el lenguaje natural, pero de una forma estructurada. Una plantilla es una tabla con una serie de campos y una estructura predefinida que el equipo de desarrollo va cumplimentando, usando para ello el lenguaje del usuario. Las plantillas eliminan parte de la ambigüedad del lenguaje natural al estructurar la información; cuanto más estructurada sea ésta, menos ambigüedad ofrece. Sin embargo, si el nivel de detalle elegido es demasiado estructurado, el trabajo de rellenar las plantillas y mantenerlas puede ser demasiado tedioso.
- **Escenarios:** La técnica de los escenarios consiste en describir las características del sistema a desarrollar mediante una secuencia de pasos. La representación del escenario puede variar dependiendo del autor. Esta representación puede ser casi textual o ir encaminada hacia una representación gráfica en forma de diagramas de flujo. El análisis de los escenarios, hechos de una forma u otra, pueden ofrecer información importante sobre las necesidades funcionales del sistema.
- **Casos de uso:** Es la técnica de especificación de requerimientos más utilizada. Actualmente, se ha propuesto como técnica básica del proceso RUP. Sin embargo, son varios los autores que defienden que pueden resultar ambiguos a la hora de especificar los requerimientos, por lo que hay propuestas que los acompañan con descripciones basadas en plantillas o en diccionarios de datos que eliminen su ambigüedad.
- **Descripción formal:** Otro grupo de técnicas que merece la pena resaltar como extremo opuesto al lenguaje natural es la utilización de lenguajes formales para describir los requerimientos de un sistema. Las especificaciones algebraicas como ejemplo de técnicas de descripción formal han sido aplicadas en el mundo de la ingeniería de requerimientos desde hace años. Sin embargo, resultan muy complejas en su utilización y su comprensión por parte del cliente. El mayor inconveniente es que no favorecen la comunicación entre cliente y analista. Por el contrario, es la

representación menos ambigua de los requerimientos y la que más se presta a técnicas de verificación automatizadas.

- **Prototipos:** Son una herramienta valiosa para proporcionar un contexto en el que los usuarios puedan comprender mejor el sistema que quieren construir. Hay una amplia variedad de prototipos que van desde maquetas gráficas (*mockups*) hasta diseños de pantalla para probar las versiones del sistema. Hay una fuerte superposición con el uso de prototipos para la validación.

2.2.3 VALIDACIÓN DE REQUERIMIENTOS

Una vez que los requerimientos han sido definidos, éstos tienen que ser validados. Por medio de la actividad de validación, la especificación de requerimientos es chequeada para validar las necesidades de los usuarios y los requerimientos de los clientes. Solo algunas aproximaciones proveen técnicas para validar requerimientos. La mayoría definen una guía de cómo los desarrolladores y clientes deben revisar la especificación de los requerimientos para identificar errores e inconsistencias. A continuación se enumeran algunas técnicas apropiadas para la validación de requerimientos:

- *Review and Walk-through*
- Auditoría
- Matriz de trazabilidad
- Validación mediante prototipos

2.3 INGENIERÍA DE REQUERIMIENTOS EN LAS METODOLOGÍAS WEB

El desarrollo de aplicaciones WEB tiene algunas características que difieren del desarrollo de otro tipo de aplicaciones. Por un lado, hay una gran variedad de participantes tomando parte del proceso de desarrollo: analistas, usuarios, clientes, diseñadores gráficos, expertos en marketing, multimedia, seguridad, etc. Por otro lado, las principales características de estos sistemas son la navegación, las interfaces y la posibilidad de realizar personalizaciones. La estructura de la navegación requiere una guía intuitiva para evitar que el usuario se pierda en el espacio navegacional. El diseño de la UI a menudo tiene que tener en cuenta aspectos de marketing y multimedia. Estos aspectos especiales no solo tienen que ser tenidos en cuenta y contemplados en la etapa de diseño, sino que deben ser contemplados ya desde la especificación de requerimientos.

En este capítulo se muestran, muy brevemente, aquellas aproximaciones que proponen técnicas y modelos para tratar con requerimientos. Existen muchas metodologías WEB que no son incluidas, dado que hacemos hincapié en la etapa de requerimientos de cada metodología.

La mayoría de las metodologías analizadas y comparadas tienen su propia clasificación de requerimientos. Sin embargo, la terminología utilizada por estas metodologías no es siempre la misma. A continuación, se muestra una clasificación de requerimientos para aplicaciones WEB para poder hacer una descripción de cada metodología que, posteriormente, pueda compararse con las otras. Está basada en el estado de arte de las metodologías WEB.

- **Requerimientos funcionales**: capacidades que el sistema debe contener para poder solucionar el problema. Pueden ser subclasificados en:
    - Requerimientos de datos
    - Requerimientos de interfaz
    - Requerimientos de navegación

- Requerimientos de perfil de usuarios
- Requerimientos transaccionales

- **Requerimientos no funcionales**: Estos actúan como restricciones que el sistema debe cumplir: portabilidad, reuso, usabilidad, rendimiento, etc.

En la próxima sección, incluimos solamente algunas de aquellas metodologías que contienen la etapa de manejo de requerimientos en sus correspondientes procesos de desarrollo.

2.3.1 PROPUESTAS WEB

2.3.1.1 WSDM (*WEB SITE DESIGN METHOD)*

WSDM [DTL97] es una propuesta para el desarrollo de sitios WEB, en la que el sistema se define basándose en los grupos de usuarios. Su proceso de desarrollo se divide en cuatro fases: modelo de usuario, diseño conceptual, diseño de la implementación e implementación. La fase que más repercusión tiene es la primera, en la que se intenta detectar los perfiles de usuarios para los cuales se construye la aplicación. Para ello, se deben realizar dos tareas:

- Clasificación de usuarios: en este paso se deben identificar y clasificar a los usuarios que van a hacer uso del sistema. Para ello, WSDM propone el estudio del entorno de la organización donde se vaya a implantar el sistema y los procesos que se vayan a generar, y la descripción de las relaciones entre usuarios y actividades que realizan estos usuarios. Para la representación gráfica de estas relaciones, WSDM propone una especie de mapas de conceptos de roles y actividades.

- Descripción de los grupos de usuarios: en esta segunda etapa, se describen con mayor detalle los grupos de usuarios detectados en la etapa anterior. Para ello, se debe elaborar un diccionario de datos, en principio con formato libre, en el que se indican los requerimientos de almacenamiento de información, requerimientos funcionales y de seguridad para cada grupo de usuarios.

El resto de las fases del proceso de WSDM se hacen a partir de la clasificación de usuarios que se realiza en esta primera etapa.

2.3.1.2 OOHDM (*OBJECT ORIENTED HYPERMEDIA DESIGN MODEL)*

OOHDM es un método ampliamente aceptado para el desarrollo de aplicaciones WEB [SR98], cuyas primeras versiones se centraban en el diseño y no incluían ingeniería de requerimientos. El proceso de OOHDM está compuesto por cuatro etapas que arrojan los siguientes resultados:

- El modelo conceptual, representado como un modelo de clases. Éste muestra una visión estática del sistema.
- El modelo navegacional consiste en un diagrama de navegación de clases y un diagrama de estructura de navegación. El primero muestra la navegación estática del sistema y el segundo muestra una extensión del diagrama de clases que incluye estructuras de acceso y contextos de navegación.
- El modelo de interfaz abstracta es desarrollado usando una técnica especial llamada ADV (*Abstract Data View*)
- La implementación consiste en el código implementado basado en los modelos previos.

En sus comienzos, no contemplaba la fase de captura y definición de requerimientos, pero actualmente propone el uso de *User Interaction Diagrams* (UIDs) [VSS00]. Esta propuesta parte de los casos de uso, que se considera una técnica muy difundida, ampliamente aceptada y fácilmente entendible por los usuarios y clientes no expertos, pero que resulta ambigua para el equipo de desarrollo en fases posteriores del ciclo de vida. Igualmente, resalta la necesidad de empezar el diseño del sistema, especialmente en los entornos WEB, teniendo un conocimiento claro y amplio de las necesidades de interacción, es decir, de la forma en la que el usuario va a comunicarse con el sistema.

Partiendo de estas dos premisas, OOHDM propone que la comunicación con el usuario se realice utilizando los casos de uso, y a partir de ellos los analistas elaboren los UIDs. Estos UIDs son modelos gráficos que representan la interacción entre el usuario y el sistema, sin considerar aspectos específicos de la interfaz. El proceso de transformación de un caso de uso a un UIDs es descripto detalladamente en la propuesta.

OOHDM centra el desarrollo de un sistema de información WEB en torno al modelo conceptual de clases. Este diagrama debe surgir de los requerimientos que se definan del sistema, pero los casos de uso resultan demasiado ambiguos para ello. Así, propone refinar el proceso de desarrollo descrito en UML, de forma que de los casos de uso se generen los UIDs que concreticen más la definición de los requerimientos para, a partir de ellos, obtener el diagrama conceptual. En algunos de los primeros trabajos, OOHDM propone la descripción de escenarios en forma textual y gráfica para cada tipo de usuario como etapa previa al diseño de la navegación.

2.3.1.3 UWE *(UML-BASED WEB ENGINEERING)*

UML-Based Web Engineering (UWE) es una propuesta metodológica basada en el Proceso Unificado y en UML [BRJ99] para el desarrollo de aplicaciones WEB.

UWE cubre todo el ciclo de vida de este tipo de aplicaciones, y centra además su atención en aplicaciones personalizadas (adaptativas). Interesa principalmente analizar la propuesta de captura de requerimientos de UWE. Esta metodología distingue entre la tarea de obtener requerimientos, definirlos y validarlos. El resultado final de la captura de requerimientos en UWE es un modelo de casos de uso acompañado de documentación que describe los usuarios del sistema, las reglas de adaptación, los casos de uso y la interfaz.

UWE clasifica los requerimientos en dos grandes grupos: funcionales y no funcionales.

Los requerimientos funcionales tratados por UWE son:

- requerimientos relacionados con el contenido
- requerimientos relacionados con la estructura
- requerimientos relacionados con la presentación
- requerimientos relacionados con la adaptación
- requerimientos relacionados con los usuarios

Además, UWE propone como técnicas apropiadas para la captura de los requerimientos de un sistema WEB las entrevistas, los cuestionarios, los *checklists*, los casos de uso, los escenarios y el glosario para la definición de requerimientos. Para la validación propone *walk-throughs*, auditorías y prototipos.

2.3.1.4 WEBML (*WEB MODELING LANGUAGE)*

El lenguaje de modelado WEB (*WebML*) es un lenguaje de alto nivel para la especificación de aplicaciones de hipermedia. *WebML* sigue dos estilos: relación de entidades y UML, y ofrece así una notación propietaria y una representación gráfica que usa sintaxis de UML. Esta notación está complementada con un conjunto de actividades a ser realizadas para el

desarrollo de las aplicaciones WEB, tales como especificación de requerimientos, diseño de datos y diseño de hipertexto [CFBB03].

La metodología se centra en la recolección y especificación de requerimientos. Propone el uso de técnicas tales como entrevistas y análisis de documentación. La recolección de requerimientos comienza con la identificación de los usuarios y los roles necesarios. También son recolectados requerimientos funcionales, no funcionales, así como los de datos. Algo a tener en cuenta es que la navegación y los requerimientos específicos de hipertexto no son tratados por separado.

La especificación de requerimientos (llamada análisis de requerimientos) consiste en la clásica especificación de casos de uso con el adicional de una descripción textual semiestructurada. El uso de diagramas de actividad es propuesto para *workflows* complejos de casos de uso. Una descripción basada en plantillas y *mockups (sketches)* son sugeridos para la especificación de la vista del sitio (guías de estilo). Finalmente, se proponen tests de aceptación, principalmente para verificar requerimientos no funcionales.

2.3.1.5 NDT (*NAVIGATIONAL DEVELOPMENT TECHNIQUES)*

NDT (*Navigational Development Techniques*) [ETM02] es una técnica para especificar, analizar y diseñar el aspecto de la navegación en aplicaciones WEB. El flujo de especificación de requerimientos de NDT comienza con la fase de captura de requerimientos y estudio del entorno. Para ello, plantea el uso de técnicas como las entrevistas o el *brainstorming* y JAD. Tras esta fase, se propone la definición de los objetivos del sistema. A partir de estos objetivos, el proceso continúa definiendo los requerimientos que el sistema debe cumplir para cubrir los objetivos marcados. NDT clasifica los requerimientos en:

- requerimientos de almacenamiento de información
- requerimientos de actores

- requerimientos funcionales
- requerimientos no funcionales
- requerimientos de interacción, representados mediante:
    - Frases que recogen cómo se va a recuperar la información del sistema; utilizan un lenguaje especial denominado BNL *(Bounded Natural Language)*
    - Prototipos de visualización que representan la navegación del sistema, la visualización de los datos y la interacción con el usuario.

Todo el proceso de definición, captura de requerimientos y objetivos que propone NDT se basa principalmente en plantillas o patrones, pero también hace uso de otras técnicas de definición de requerimientos, como son los casos de uso y los glosarios. La propuesta ofrece una plantilla para cada tipo de requerimiento, lo que permite describir los requerimientos y objetivos de una forma estructurada y detallada. Algunos de los campos de los patrones son cerrados, por lo tanto solo pueden tomar valores predeterminados. Estos campos permiten que en el resto del proceso del ciclo de vida de NDT se puedan conseguir resultados de forma sistemática. El flujo de trabajo de especificación de requerimientos termina proponiendo la revisión del catálogo de requerimientos y el desarrollo de una matriz de trazabilidad que permite evaluar si todos los objetivos han sido cubiertos en la especificación.

La propuesta viene acompañada de una herramienta *CASE*, NDT-*Tool*, que facilita la cumplimentación de los patrones y permite automatizar el proceso de consecución de resultados.

2.3.1.6 OOWS (*OBJECT ORIENTED WEB-SOLUTIONS)*

OOWS [PAF01] es una metodología que se centra en el desarrollo de sistemas de información hipermedia y aplicaciones de comercio electrónico en

ambientes WEB. OOWS está basada en los modelos conceptuales de OO-*Method* [GCP00] y propone la utilización del diseño navegacional, junto con el modelado conceptual, como punto de partida para la generación de código de aplicaciones WEB.

OOWS propone un proceso de desarrollo de dos grandes pasos: especificación del sistema y desarrollo de la solución. Dentro de la especificación del sistema,, se contempla una etapa de recolección de requerimientos y la realización de los siguientes modelos: modelo de dominio, modelo dinámico, modelo funcional y modelo de navegación. Como parte del desarrollo de la solución, se contempla el desarrollo de la aplicación en tres capas: datos, lógica de negocio y presentación.

- La etapa de recolección de requerimientos propone la especificación del comportamiento del sistema a través de diagramas de casos de uso.
- En el modelo de objetos se define la estructura estática (clases, con sus atributos, operaciones, y relaciones entre ellas) del dominio de la aplicación. Este modelo se representa por medio de un diagrama de clases UML.
- El modelo dinámico se especifica a través de un diagrama de transición de estado y describe la “vida” de los objetos, es decir, representa el comportamiento de cada clase en el sistema.
- El modelo funcional captura la semántica asociada a los cambios de estado de los objetos motivado por la ocurrencia de servicios.
- El modelo navegacional se compone de un mapa navegacional que se representa como un grafo dirigido. Los nodos del grafo son contextos navegacionales y los enlaces representan la navegación de un contexto a otro. Cada contexto navegacional representa un punto de vista de los usuarios sobre los objetos del modelo de objetos.

Como su nombre lo indica, OOWS utiliza los principios del paradigma orientado a objetos para el desarrollo de aplicaciones WEB. En cuanto al

proceso de desarrollo del hipertexto, se puede decir que OOWS utiliza un enfoque similar al OOHDM, orientado también a objetos. Así, de manera similar a OOHDM, OOWS propone construir el hipertexto de la aplicación WEB en base a los contextos navegacionales, los cuales representan las clases del dominio de la aplicación y las relaciones entre ellas.

OOWS utiliza notación UML para la representación de sus modelos, aunque también propone la realización de diagramas que utilizan otra notación, como el diagrama de *ConcurTaskTrees* [PMM97], que sirve para realizar taxonomías de tareas [VFP05]. Hasta donde se conoce, los autores de OOWS no han definido un perfil UML que reúna los elementos de modelado propuestos por la metodología.

2.4 COMPARACIÓN

Una vez enunciadas las propuestas, se presenta en esta sección una comparación de éstas por medio de una tabla (Fig. 2). Ésta presenta las distintas técnicas enunciadas en la Sección 2.2.2. Debido al alcance de esta tesis, solo se compara la fase de especificación de requerimientos.

Se ve en la Fig. 3 cuáles son las técnicas que soportan transformaciones o derivaciones, y en la Fig. 4 se ven los tipos de transformaciones de estas técnicas.

	WSDM	OOHDM	UWE	WebML	NDT	OOWS
Lenguaje natural	✓			✓		
Glosarios			✓		✓	
Patrones/Plantillas				✓	✓	
Escenarios			✓			✓
Casos de uso		✓	✓	✓	✓	✓
Lenguaje formal						
Prototipos						
Otras técnicas	Diagrama de tareas	UIDs	Diagrama de actividades	Diagrama de actividades	Frases BNL	FRT Diagrama de tareas

Fig. 2. Técnicas de especificación de requerimientos

	WSDM	OOHDM	UWE	WebML	NDT	OOWS
Derivación de aplicación o modelos			✓		✓	✓
Herramienta CASE			MagicUWE		NDT-Suite	OOWS-Suite

Fig. 3. Capacidades de derivación o transformación

	UWE	NDT	OOWS
Transformación modelo a modelo	✓	✓	✓
Transformación modelo a texto		✓	

Fig. 4. Transformaciones

En la próxima sección, se presentarán las conclusiones obtenidas del análisis y de la comparación detallados en esta sección.

2.5 CONCLUSIONES

Con respecto a la especificación de requerimientos, se puede observar que es el aspecto central del tratamiento de requerimientos para todas las propuestas. Se puede concluir que existe una tendencia a usar la técnica de casos de uso como base. Sin embargo, ante este escenario, hay dos opiniones encontradas. Algunas propuestas consideran los casos de uso como una técnica óptima para representar los requerimientos, tal es el caso de UWE. Sin embargo, hay otras como OOHDM o NDT que, aunque indican que es una buena técnica, resaltan que es ambigua y que es necesario obtener modelos más concretos para hacer más sistemático el resto del proceso de ciclo de vida.

Con respecto a las capacidades de las propuestas para realizar derivaciones o transformaciones, concluimos que son pocas las propuestas que tienen esta posibilidad; en la Fig. 3, podemos ver claramente que solo tres de ellas son aptas. Por otro lado, también se observa que hay muy pocas herramientas *CASE* que den soporte a estas propuestas.

Con respecto a qué tipo de transformaciones soportan las propuestas enunciadas, se observa que la mayoría se centra en transformaciones de modelo a modelo (Fig. 4). Generalmente estas transformaciones se empiezan a aplicar en la etapa de diseño.

NDT es la única propuesta que soporta transformaciones modelo a texto. Una vez validados los requerimientos, el proceso de NDT propone generar tres modelos: el **modelo conceptual**, que representa mediante un diagrama de clases la estructura estática del sistema; el **modelo de navegación**, que representa mediante un conjunto de diagramas de clases la forma en que se podrá navegar en el sistema; y el **modelo de interfaz abstracta**, que mediante

un conjunto de prototipos evaluables permitirá mostrar cómo se va a interactuar con el sistema. La característica más destacable del proceso propuesto por NDT es que el paso de la especificación de requerimientos a estos modelos se hace de una manera sistemática e independiente. Además, de la especificación de requerimientos se puede derivar en el documento de requerimientos del sistema, y de los modelos obtenidos de la especificación de requerimientos se puede derivar en el documento de análisis del sistema y prototipos evaluables.

CAPÍTULO 3

BACKGROUND

Muchos estudios de casos reales en la industria del software muestran la importancia de los requerimientos en el desarrollo de aplicaciones WEB. Desafortunadamente, en este tipo de aplicaciones, los requerimientos generalmente son descriptos en documentos informales por medio de casos de uso. Estos documentos carecen de las particularidades de la WEB, tales como la interacción y la navegación. El hecho de que los integrantes de los proyectos tienen roles y perfiles diferentes (clientes, usuarios, diseñadores, programadores, etc.), sumado a que los requerimientos cambian constantemente, hacen que la situación sea aún peor.

La rápida evolución de las aplicaciones WEB plantea dificultades adicionales en contra del proceso de pruebas de la aplicación. Esto es debido al continuo cambio de la especificación de los requerimientos. Además, las técnicas o herramientas de requerimientos deben ser lo suficientemente expresivas y sencillas de entender por cualquier miembro del equipo, con la finalidad de que puedan ser validados antes de empezar la etapa de implementación. Otro requisito es que la aplicación tiene que ser validada para verificar que los requerimientos especificados fueron implementados correctamente y, más importante aún, que los requerimientos previos siguen funcionando como se espera.

En el contexto de la ingeniería WEB dirigida por modelos [MDWE], las anteriores observaciones no han sido tomadas demasiado en cuenta, por lo que las aplicaciones WEB desarrolladas con estas metodologías sufren problemas

tales como: requerimientos desactualizados, la imposibilidad de aseverar que la aplicación cumple con los requerimientos, y la complejidad que conlleva el manejo de la rápida evolución de los requerimientos. Para hacer frente a todos estos problemas, se ha desarrollado *WebSpec* [RLRG1], un lenguaje de propósito múltiple utilizado para capturar la interacción, la navegación y las características de la UI (interfaz gráfica de usuario).

En este capítulo, se presenta *WebSpec*, sus conceptos y sintaxis (Sección 3.1), además se muestra cómo *WebSpec* es utilizado en diferentes actividades del ciclo de desarrollo (Sección 3.2), y finalmente se muestra la herramienta de soporte para *WebSpec* (Sección 3.3).

3.1 *WEBSPEC* (UN DSL PARA CAPTURAR REQUERIMIENTOS WEB INTERACTIVOS)

Las aplicaciones WEB tienden a cambiar con demasiada rapidez, y es muy complicado para los equipos de desarrollo poder llevar a cabo esos cambios rápidamente. Las metodologías ágiles, por su esencia en sí y mediante la continua interacción de los integrantes, mejoran levemente este aspecto. La recolección de estos cambios generalmente se hace de una manera informal, por lo cual es casi imposible validar si éstos se implementaron correctamente. A menudo, los equipos de desarrollo agregan tests no solo para validar los requerimientos, sino también para guiar las decisiones de diseño (*Test Driven Development* [BTDD2]). Cuando la aplicación evoluciona, y el número de requerimientos implementados crece, los tests son de suma importancia para poder verificar que los requerimientos previos siguen siendo correctos (*Regression Tests*).

Para poder capturar requerimientos WEB, los investigadores han tomado prestado los casos de uso e historias de usuarios de la ingeniería de software tradicional para poder adaptarlos a la ingeniería de software WEB. Estas

herramientas permiten definir los requerimientos de una forma semiestructurada e informal que permite una interacción apropiada y flexible con los clientes. Sin embargo, éstos no contribuyen a especificar detalles de la UI que son esenciales en las aplicaciones WEB y, por lo tanto, la validación de estos aspectos en las aplicaciones WEB se hace manualmente. Como consecuencia, la validación de requerimientos solo se realiza sobre los últimos requerimientos implementados, por ende, los efectos que estos nuevos requerimientos puedan llegar a tener sobre los requerimientos previos solo se descubren una vez que el usuario detecta un error en la aplicación.

Por otro lado, hay lenguajes más formales que ayudan a especificar los requerimientos interactivos con mayor precisión, lo que hace más fácil su implementación para el equipo de desarrollo, ya que suelen proporcionar algún tipo de derivación automática de la estructura de la base de la aplicación (por ejemplo, la topología de las páginas y los enlaces entre ellas). Sin embargo, en general, no presentan derivación automática de tests, y los que están relacionados con algún enfoque de desarrollo dirigido por modelos (MDWE) tienden a estar estrechamente ligados a las técnicas de modelado del enfoque. Para empeorar las cosas, muchas veces son demasiado abstractos o complejos para ser usados o entendidos por los clientes y, por lo tanto, poco realistas para ser utilizados en proyectos de la vida real.

Para hacer frente a estos problemas y, a la vez, conservar las ventajas de los lenguajes mencionados, se ha desarrollado *WebSpec*. *WebSpec* es un lenguaje visual que tiene soporte para simular la aplicación y que ayuda a los clientes a visualizar los requerimientos antes de su implementación. La validación de los requerimientos se realiza automáticamente mediante la ejecución de una serie de tests obtenidos a partir de la especificación de requerimientos, que es independiente de la tecnología con la cual se implemente la aplicación, ya que se basa en los navegadores WEB y no en la tecnología utilizada para desarrollar la aplicación. Al igual que cualquier lenguaje formal, también proporciona derivación de algunas partes de la aplicación a una

tecnología particular (GWT, *Seaside*, etc), todo integrado en su herramienta de soporte, el *plugin* de *Eclipse WebSpec*.

WebSpec es un lenguaje visual específico de dominio que permite especificar la navegación, interacción, y los requerimientos de UI WEB. La herramienta principal para la especificación de requerimientos es el diagrama *WebSpec* (Sección 3.1.1), que puede contener interacciones (Sección 3.1.2), navegaciones y comportamientos ricos (Sección 3.1.3). Como una de las principales motivaciones del lenguaje es la derivación automática de tests, se tomó prestada la idea de generación para especificar las propiedades que la aplicación debe satisfacer. Por ejemplo, cualesquiera de las siguientes propiedades: “El precio de un producto debe ser un número positivo” o “Un nombre de usuario válido es una cadena de longitud entre 8 y 16 caracteres alfanuméricos” se puede especificar mediante un generador. Un generador (Sección 3.1.4) proporciona una forma simple y reutilizable para describir un conjunto de datos (por extensión o comprensión); puede ser interpretado como una función que devuelve un elemento aleatorio de un conjunto especificado. Por ejemplo, un generador de cadenas configurado con una longitud mínima de 8 y máxima de 16 se podría utilizar para obtener los nombres de usuario válidos para el caso anteriormente mencionado (por ejemplo, “administrador”). Finalmente, los diagramas de *WebSpec* pueden estar compuestos (Sección 3.1.5) para hacer frente a la complejidad y, al mismo tiempo, para permitir la reutilización de requerimientos.

WebSpec está formalmente definido en el metamodelo que se muestra en la Fig. 5. Por cuestiones de espacio, evitamos las jerarquías de *Expression* y *Widget*. Un diagrama (instancia de la clase *Diagram*) consta de interacciones (instancias de la clase *Interaction*) y transiciones ya sea de navegación o comportamiento rico (instancias de la clase *Navigation* y *RichBehavior*). Una instancia de la clase *Interaction* conoce su nombre, sus transiciones y su maqueta de UI asociada (*mockup*). Una instancia de *Transition* (*Navigation* o *RichBehavior*) conoce su origen, destino, sus precondiciones, y la secuencia de acciones que hacen posible la transición. Por último, una instancia de la clase

Interaction conoce su contenedor de *widgets*, que puede contener muchas instancias de *AbstractWidget* (*Widget* o *Container*). Cada *widget* también puede estar asociado con su maqueta de UI a través de su atributo de ubicación.

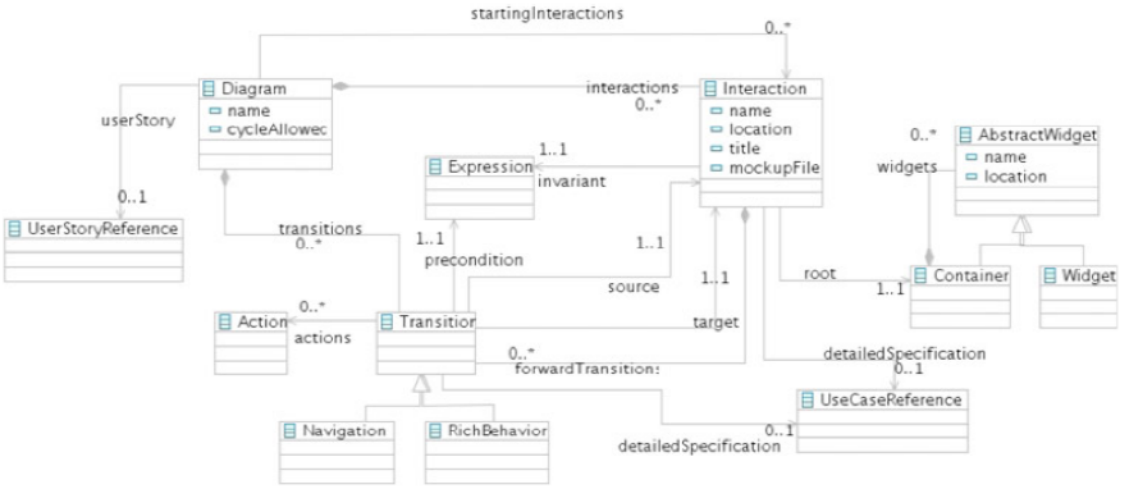


Fig. 5. Metamodelo simplificado de *WebSpec*

3.1.1 DIAGRAMAS DE *WEBSPEC*

Un diagrama *WebSpec* define un conjunto de escenarios que la aplicación WEB debe satisfacer. Éstos contienen dos elementos principales: interacciones y transiciones; las transiciones pueden ser navegaciones o comportamientos ricos. Las interacciones representan puntos donde el usuario puede interactuar con la aplicación, y las transiciones representan un movimiento de un punto de interacción a otro. Por lo tanto, un diagrama *WebSpec* puede ser visto como un grafo donde las interacciones son los nodos, y las transiciones representan las aristas. Un escenario es representado como una secuencia de interacciones y transiciones, por ejemplo: <interacción1, navegación1, interacción2, comportamientoRico1, interacción2> define un posible camino de interacción entre el usuario y la aplicación WEB.

La Fig. 6 muestra un diagrama *WebSpec* para nuestra historia de usuario de ejemplo: “Como cliente, quiero poder entrar y salir de la aplicación con mi nombre de usuario y contraseña”. El diagrama es construido en forma iterativa entre el cliente y el analista a lo largo de varias reuniones. Debido a que el uso de *WebSpec* no está atado a una metodología en particular, podemos utilizar las reuniones de larga duración típicas de métodos unificados o las reuniones cortas típicas de las metodologías ágiles. La construcción del diagrama puede ser mejorada al utilizar *mockups* y simular la aplicación (como se muestra en los próximas secciones); sin embargo, se espera que, con un poco de entrenamiento, el cliente sea capaz de construir los diagramas por su cuenta. Como ejemplo, el diagrama de la Fig. 6 define los caminos de navegación que el usuario debe seguir desde la página de inicio (*Login*) a la página principal (*HAlerts*), y luego a la página de inicio (debido a su salida de la aplicación).

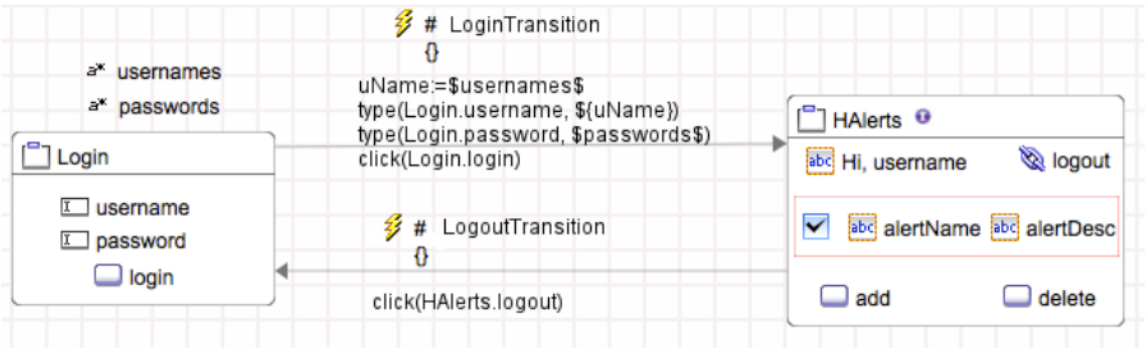


Fig. 6. Diagrama *WebSpec* para el escenario de entrada y salida de la aplicación

El conjunto de escenarios que un diagrama *WebSpec* especifica es obtenido al recorrer el diagrama y al usar un algoritmo de búsqueda (DFS – *Depth-First Search*). El algoritmo comienza por un conjunto de nodos especiales llamados nodos iniciales y sigue los lados (transiciones) del grafo (diagrama). Generalmente, uno o más diagramas podrían estar relacionados con la misma historia de usuario para especificar escenarios concretos que la aplicación WEB debe cumplir. En las siguientes dos secciones se explica detalladamente el contenido de los diagramas (Interacciones y Navegaciones).

3.1.2 INTERACCIÓN

Una interacción representa un punto donde el usuario interactúa con la aplicación mediante el uso de sus elementos de UI (*widgets*). Formalmente, representan el estado de una página WEB, la cual ha sido cargada inicialmente, o cuando ha cambiado como consecuencia de un comportamiento rico. Las interacciones poseen un nombre (único por diagrama) y pueden contener *widgets* tales como: etiquetas, listas, botones, cajas de selección y paneles. Los *widgets* definen el contenido (información) mostrado por la interacción. Para permitir la composición de *widgets,* existen dos clases: *ListPanel* y *Panel*. El *ListPanel* representa una repetición de los elementos que contiene, y el *Panel* define un contenedor simple de *widgets*. Las interacciones están representadas gráficamente por un rectángulo con los bordes redondeados (Fig. 7) que contiene el nombre de la interacción y sus *widgets*. Un diagrama *WebSpec* debe tener al menos una interacción inicial, la cual se encuentra representada con bordes punteados. Para especificar qué propiedades debe satisfacer la aplicación, hacemos uso de invariantes (expresiones booleanas) en las interacciones del diagrama. Cada interacción (implícitamente o explícitamente) define un invariante que especifica las propiedades que deben ser satisfechas en los escenarios especificados por el diagrama (en el caso de que no se defina uno explícitamente, se asume que el invariante es verdadero).

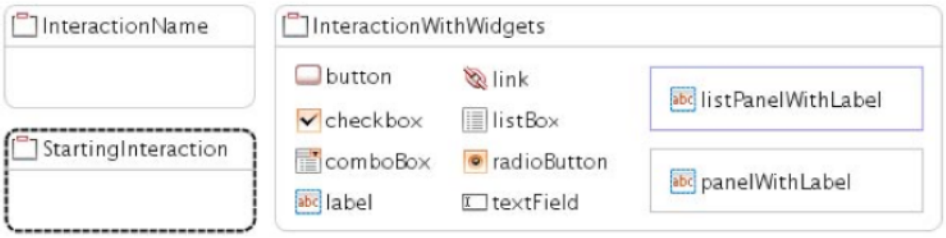


Fig. 7. Interacciones de *WebSpec*

Las expresiones booleanas se refieren a los siguientes elementos:

- Propiedades de los *widgets*: Cualquier propiedad de un *widget* que está contenida en la interacción. Por ejemplo, *Login.username.text* se refiere al valor textual del *widget username*.
- Variables: Cuando se necesita acceder a un valor o a una propiedad de una interacción anterior del escenario, se deben guardar en variables, por ejemplo, *username*:="mcataldo" o *username:=Login.username.text*. Accedemos al valor de la variables con la sintaxis *${nombre de la variable}*.
- Generadores: Como se mostrará en la Sección 3.1.4, los generadores pueden ser referidos con la sintaxis *$nombre del generador$*, por ejemplo, *username*:=$*usernames*$.
- Expresiones compuestas: Es posible componer expresiones mediante (&&), (||), (→), (!).

En la Fig. 6, vemos que la interacción *HAlerts* define un invariante (marcado con un ícono que tiene la letra I junto al nombre de la interacción): *HAlerts.username.text* = ${*uName*} que indica que el texto de la etiqueta *HAlerts.username* tiene que ser igual al valor de la variable *uName*.

3.1.3 NAVEGACIÓN

Generalmente, el comportamiento de las aplicaciones WEB se muestra por la navegación de una página hacia otra o por cambios locales en una página específica que no implican la navegación hacia otra página. Estos cambios producidos en el navegador WEB, ya sean navegación de una página a otra o cambios en una página, son percibidos por el usuario por cambios en el historial

de navegación o en la UI de la pagina, respectivamente. A estos cambios se los llama comportamientos interactivos (Sección 3.1.3.1).

Por otro lado, hay comportamientos que no son interactivos (Sección 3.1.3.2), los cuales no son percibidos directamente por el usuario. Estos son iniciados por la navegación de una página hacia otra. Ejemplos de estos comportamientos son: enviar un correo electrónico, realizar un cobro a una tarjeta de crédito, realizar la autenticación de usuario por medio de un sistema externo como Facebook o Google, etc.

3.1.3.1 COMPORTAMIENTO INTERACTIVO

Cuando el usuario navega de una página hacia otra, un nuevo elemento en el historial de navegación es agregado, de esta manera se le permite al usuario volver a la página anterior. Durante la obtención de requerimientos, esto es percibido, dado que el cliente expresa “En esta página querría que los usuarios puedan volver a la página anterior”.

En *WebSpec*, una navegación está representada gráficamente (Fig. 8) con flechas grises, mientras que su nombre, precondición y las acciones que la activan están mostrados como etiquetas sobre ella. En particular, su nombre aparece con el carácter “#” como prefijo, su precondición entre {}, y las acciones en las siguientes líneas. Debemos resaltar que la idea detrás de las acciones de una transición (sean estas navegaciones o comportamientos ricos) es que su ejecución produce una transición entre las interacciones y no al revés. Una transición debe ser entendida como: “Si la precondición se satisface, y el usuario ejecuta la secuencia de acciones, la aplicación debe transitar a la interacción destino”.

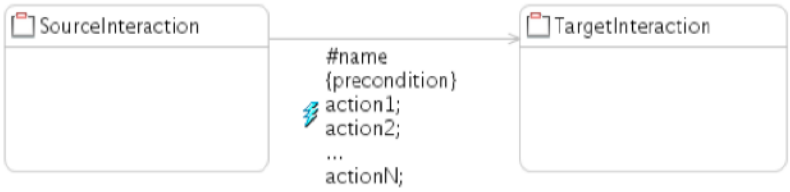


Fig. 8. Una navegación en *WebSpec*

Una navegación de una interacción a otra puede ser activada si su precondición se satisface por medio de la ejecución de su secuencia de acciones tales como: presionar en un botón, agregar texto a un campo, etc. Así como los invariantes, las precondiciones pueden referir a variables definidas previamente en el diagrama. Las acciones están escritas de acuerdo a la siguiente sintaxis: *var := expr | actionName(arg1,... argn)*.

Los links de navegación tradicionales se representan sin precondición (implícitamente existe una precondición por defecto con valor verdadero) y con una sola acción: presionar (un *widget* enlace) como se muestra en la interacción *AddAlert* (Fig. 9).

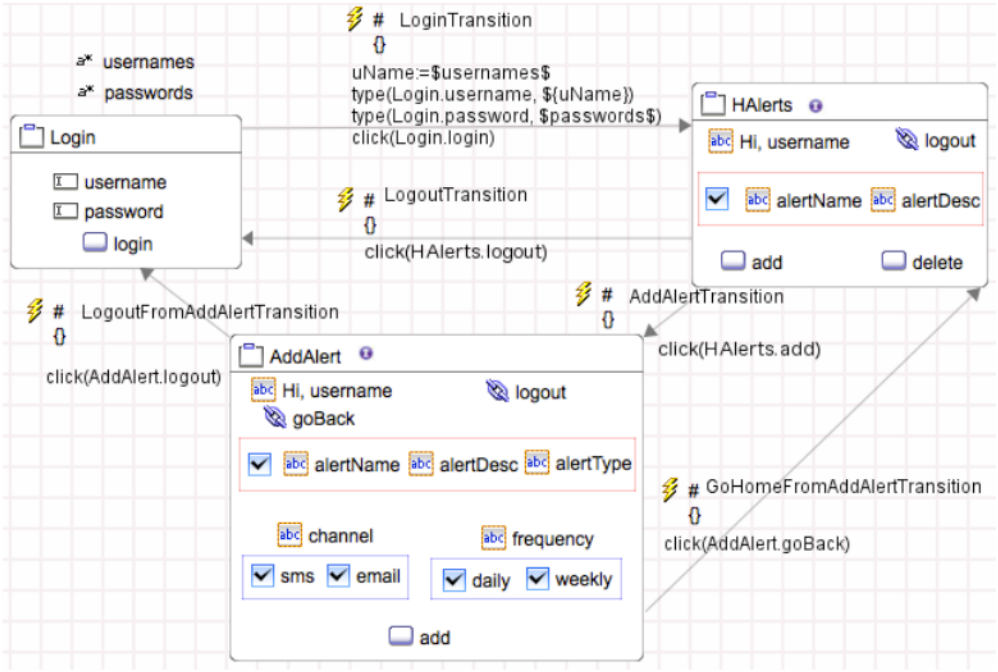


Fig. 9. Interacción *AddAlert* y navegación de retorno a la página de inicio

Un ejemplo más complejo de una secuencia de acciones se presenta a continuación *(#LoginTransition*, extraído de la Fig. 9):

(1) *uName := $usernames$*
(2) *type(Login.username, ${uName})*
(3) *type(Login.password, $passwords$)*
(4) *click(Login.login)*

La primera sentencia asigna el valor generado por el generador "usernames" (denotado entre $) a la variable "uName" (para un uso posterior). En la segunda sentencia, el contenido de la variable "uName" es escrito en el *widget* "username" (campo de texto); posteriormente, en la tercera sentencia, se escribe el valor generado por el generador "passwords" en el *widget* "password" (campo de texto del tipo contraseña). Finalmente, en la cuarta sentencia, se presiona en el botón "login".

Por otro lado, la aplicación puede cambiar el estado de la UI (interfaz gráfica de usuario) como consecuencia de algunas acciones ejecutadas por el usuario. Por ejemplo, cuando el mouse está "sobre" un *widget*, más información podría ser mostrada en un *pop-up*, o cuando se está escribiendo texto en un campo de texto, pueden aparecer opciones como en un campo de autocompletado. Estos cambios locales son usuales en las llamadas aplicaciones ricas de Internet (RIA), y es común hoy en día que los clientes pidan requerimientos de este estilo, tanto explícitamente ("Yo quiero un campo *auto-complete*"), o implícitamente ("Quiero que la información aparezca como cuando se busca en Amazon.com"). Estos comportamientos ricos se utilizan cada vez con mayor frecuencia en las aplicaciones WEB 2.0, pero también en las tradicionales.

En una aplicación WEB, un comportamiento rico es percibido como un cambio local en la UI que no agrega un elemento nuevo al historial de navegación del explorador WEB. Este tipo de comportamiento posee las mismas

propiedades de una navegación (nombre, precondición y acciones). Para especificarlo en *WebSpec*, utilizamos flechas rojas con líneas punteadas (Fig. 10).

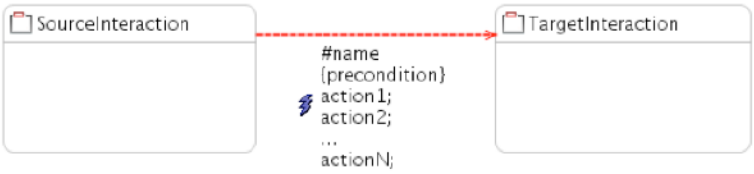


Fig. 10. Un comportamiento rico en *WebSpec*

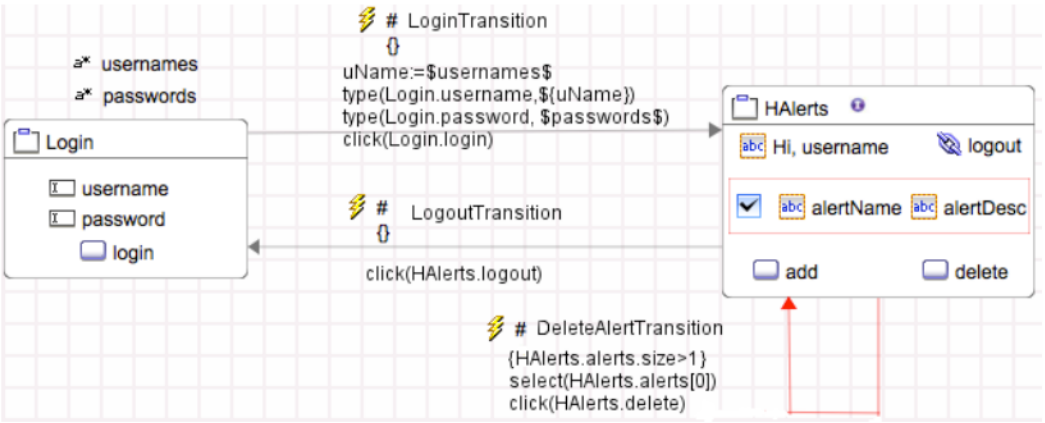


Fig. 11. Comportamiento rico *DeleteAlertTransition*

3.1.3.2 COMPORTAMIENTO NO INTERACTIVO

La mayoría de los requerimientos de las aplicaciones WEB están relacionados con aspectos interactivos que pueden ser especificados con invariantes y acciones. Sin embargo, como se mencionó anteriormente, existen ciertos escenarios que podrían tener comportamientos “ocultos” (no percibidos directamente por el usuario desde el punto de vista de una interacción) que sería importante especificar.

Para capturar este tipo de requerimientos, *WebSpec* puede ser combinado con dos herramientas diferentes (que dependen del nivel de detalle necesario) para la especificación de comportamientos ocultos. Si se necesita especificar una funcionalidad simple que no requiera una compleja lógica de negocios, se podrían utilizar notas, las cuales pueden asociarse a las interacciones y transiciones. Las notas proporcionan una forma sencilla y simple de especificar detalles que no serán percibidos desde el punto de vista del usuario. En la Fig. 12 se muestra una nota en la que se especifica que la autenticación del usuario se hace a través de otro sistema, en este caso Facebook.

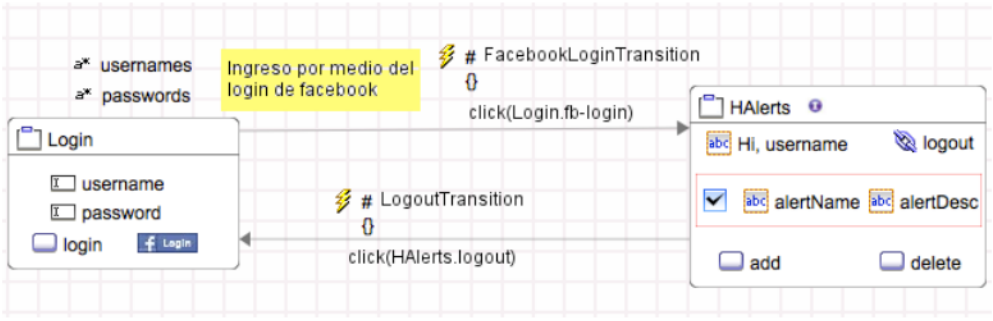


Fig. 12. Nota que explica la implementación del proceso de autenticación

Por otro lado, existen otros casos más complejos, tales como llamadas a servicios WEB, transacciones de tarjetas de crédito, etc., que no pueden ser especificados con notas. Estos casos pertenecen a alguna de las siguientes categorías:

- Complejas integraciones entre aplicaciones (WEB o no) que son difíciles de lograr y que, generalmente, contienen complejas interfaces de comunicación u otros contratos y formatos de intercambio de información. En estos casos, es aconsejable utilizar documentos detallados acerca de estos requerimientos.
- Detalles técnicos de bajo nivel, tales como información que necesita ser almacenada en archivos de *logs* como parte del proceso de negocio de la

aplicación. Generalmente, esta información queda almacenada en servidores.

- Cualquier comportamiento que no es percibido desde el punto de vista del usuario, tal como la generación de reportes PDF con información estadística sobre la actividad de los usuarios.

En todos estos casos, *WebSpec* permite asociar casos de uso con interacciones y transiciones para una mejor descripción del requerimiento (se puede ver la relación entre *Interaction* y *Transition* con *UseCaseReference* en la Fig. 5).

3.1.4 GENERADORES DE DATOS

Con *WebSpec*, tenemos la posibilidad de especificar valores generales y concretos para las propiedades de la aplicación. Una propiedad puede ser especificada con un valor concreto en uno o más escenarios que usan valores constantes en la acciones (*type(Login.username, "admin"*)) o en las invariantes (HAlerts.username="admin"). Por otro lado, a veces es necesario especificar propiedades con valores un poco mas complejos: por ejemplo, si se quisiera especificar un valor a una propiedad, y que el resultado de la lógica de validación muestre el siguiente comentario "el campo *password* debe ser superior a 10 caracteres", se debería especificar la propiedad "password" de la interacción *Login* con una cadena de caracteres menor a 10 (*type(Login.password, cadenaDeCaracteresMenorA10*)).

Para especificar valores generales de la propiedades, se puede crear el diagrama con valores concretos y luego abstraerlos con generadores. Los generadores son necesarios para mapear escenarios abstractos (aquellos sin valores concretos) con instancias de escenarios concretas (con la correspondiente distribución de datos). Este mapeo es utilizado durante la generación de tests (Sección 3.2.3) y en las simulaciones (Sección 3.3.1). Un

generador ayuda a definir cuál es el conjunto de datos válidos para los diferentes escenarios; también ayuda al equipo de desarrollo (dado que es una especificación formal del conjunto de datos) a implementar cada escenario de acuerdo con la lógica esperada.

Basándose en la idea de *QuickCheck* [CH00], los generadores son los encargados de producir la información necesaria para la especificación de los requerimientos de interacción. Un generador es una función que puede ser llamada desde las acciones de las transiciones (*type(Login.password, $passwords$*)) y generar la información necesaria. En la Fig. 11, se puede ver un ejemplo del uso de generadores para obtener un valor y asignarlo a la variable "uName" (*uName:=$usernames$*). Un generador también puede ser visto como la definición por comprensión de un conjunto, por ejemplo, el generador *usernames* = "todas las cadenas de caracteres alfanuméricas entre 5 y 10 caracteres de longitud", [*admin, superAdmin,….*].

Con el objetivo de especificar diferentes tipos de requerimientos, *WebSpec* provee una variedad de generadores similares a los que *QuickCkeck* provee. Agregar un nuevo generador en *WebSpec* no es una tarea complicada. A continuación, se describen los generadores que *WebSpec* provee:

- Uno de muchos caracteres alfabéticos: El usuario especifica un conjunto de caracteres, y el generador elige uno con distribución uniforme. Por ejemplo, si el generador es configurado con los valores "uno", "dos", "tres", el generador podría generar el valor "dos".
- Uno de muchos caracteres numéricos: Similar al generador anterior, por ejemplo, si se configura el generador con 4, 5, 6, 10.5, 3, -1, el generador podría generar el valor 10.5.
- Distribución uniforme de números: El generador se configura con los valores mínimos y máximos, y este selecciona uno en el intervalo con igual probabilidad. Por ejemplo, para el intervalo [2.6, 8.7], el generador podría generar el valor 6.

- Cadena de caracteres aleatoria: Se configura el generador con una longitud mínima y máxima de caracteres a generar, luego el generador genera un valor aleatorio con la longitud especificada en el intervalo.
- Uno de muchos arreglos: El generador se configura con un conjunto de arreglos y genera un valor con igual probabilidad de ser elegido dentro del conjunto especificado. Los arreglos se utilizan cuando hay interdependencia entre datos, por ejemplo, el arreglo de usuarios válidos que tienen nombre de usuario y contraseña: [*admin, admin*], [*user, userpass*], [*root, rootpass*], luego el generador podría generar el valor [*user, userpass*].

Cada uno de estos generadores tiene una representación visual que veremos a continuación.

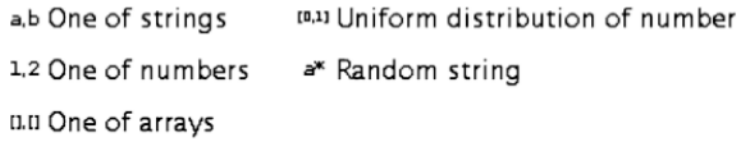


Fig. 13 Diferentes generadores de *WebSpec*

3.1.5 COMPOSICIÓN DE DIAGRAMAS

Cuando la aplicación crece, los nuevos requerimientos pueden referenciar a los requerimientos previamente descriptos. Supongamos que contamos con los siguientes requerimientos expresados con historias de usuario: “Como administrador, me gustaría poder buscar usuarios por su correo electrónico para poder bloquearlos” y “Como administrador, me gustaría poder buscar usuarios por correo electrónico para poder ver su actividad”. Ambos requerimientos hacen referencia a cierta funcionalidad del administrador con respecto a acciones que

se quieren realizar: buscar usuarios por su correo electrónico, bloquearlos y ver su actividad. Fig. 14 a y Fig. 14 b muestran los diagramas correspondientes a cada requerimiento.

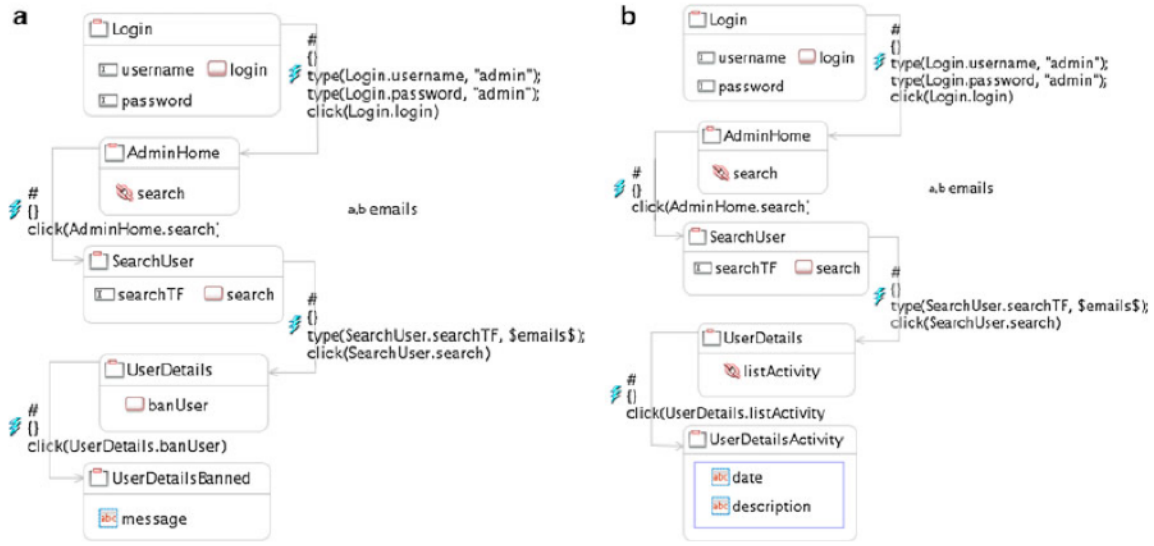


Fig. 14. a. Diagrama para bloquear usuarios, **b.** Diagrama para ver la actividad de los usuarios

Se observa que ambos diagramas tiene una secuencia de interacciones y transiciones en común que especifican las precondiciones para poder expresar el requerimiento. En este caso, la secuencia *–Login → AdminHome → SearchUser–* es común en ambos diagramas, y su principal objetivo es poder ingresar como un usuario administrador y buscar usuarios en el sistema. Las interacciones y navegaciones que están después de esta secuencia son las que expresan ambos requerimientos.

El concepto de operación se define para poder afrontar los temas de comprensión y escalabilidad; este concepto está basado en la noción de composición. En la Fig. 15, vemos la definición de la operación “LoginAsAdminAndSearchForUser”.

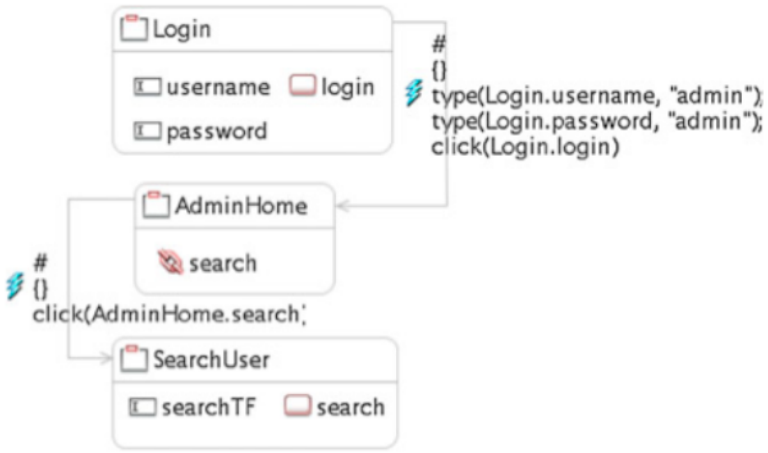


Fig. 15. Operación *LoginAsAdminAndSearchForUser.*

Como consecuencia, los diagramas de las Fig. 14 a y b pueden ser escritos de una forma más acotada como se muestra en Fig. 16 a y b Estos diagramas son la composición entre la operación *LoginAsAdminAndSearchForUser* y las secuencias de las Fig. 14 a y b.

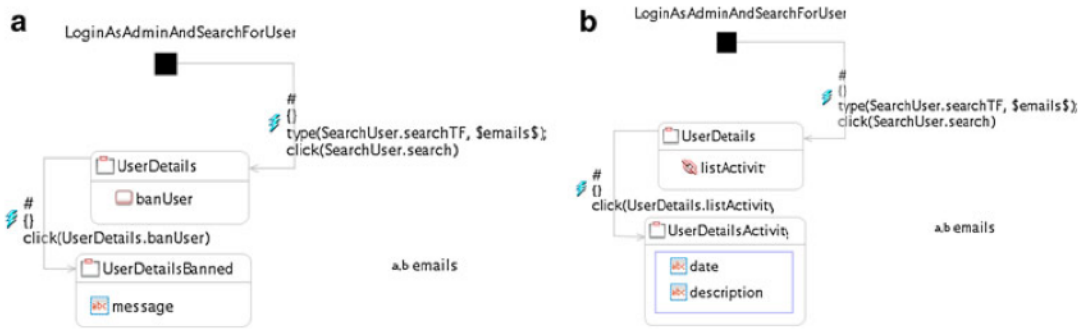


Fig. 16. a. Diagrama factorizado del bloqueo de usuario y **b.** Diagrama factorizado de la visualización de actividades de usuarios

3.2 *WEBSPEC* EN ACCIÓN

En las secciones previas, se ha mostrado el lenguaje y la forma en la que se puede especificar requerimientos interactivos en aplicaciones WEB; en esta sección, se explica cómo *WebSpec* es utilizado en el ciclo de desarrollo. A modo de introducción, se detalla cómo se utiliza en la práctica un diagrama que tiene ciclos y especifica infinitos escenarios (Sección 3.2.1). Luego, se muestran algunas características de *WebSpec*, tales como la simulación de la aplicación (Sección 3.2.2), la validación de los requerimientos (Sección 3.2.3) y los cambios en los requerimientos (Sección 3.2.4).

3.2.1 ESCENARIOS INFINITOS

Como se muestra en la Fig. 6, los diagramas de *WebSpec* pueden especificar un conjunto infinito de escenarios (*login-logout-login-logout*-…) cuando tienen ciclos. Por ejemplo, el diagrama de la Fig. 6 tiene un ciclo entre las interacciones *Login* y *HAlerts*. Al presentar ciclos en un diagrama, nos encontramos con un problema importante que solucionar con respecto a la simulación y a la validación de requerimientos. Esto se debe a que la simulación y la validación no sabrían cuando parar. En estos casos, se ha adoptado una solución pragmática: como los escenarios son infinitos, y ni la simulación ni la validación finalizarían, se redujo la cantidad infinita de caminos a una cantidad constante de ocurrencias de una interacción. Por lo tanto, un escenario puede terminar en una interacción que no tiene navegación o cuando el número de ocurrencias de la interacción alcanzó el máximo disponible para el diagrama.

3.2.2 MEJORANDO LA COMPRENSIÓN DE REQUERIMIENTOS

Con el objetivo de mejorar la etapa de obtención de requerimientos, los diagramas de *WebSpec* pueden simular la aplicación en desarrollo. La simulación es importante para reducir la diferencia en el entendimiento de un requerimiento entre los clientes y los analistas y, por lo tanto, ayuda a obtener *feedback* real de ellos. Usualmente, los clientes necesitan un determinado nivel de conocimiento para poder comprender totalmente las herramientas de captura de requerimientos. Si esto no es así, se generan problemas que son detectados cuando la aplicación se encuentra en la etapa de desarrollo.

En *WebTDD* (*WEB Test Driven Development [WTDD]*), la simulación puede ser utilizada cuando creamos los *mockups* y los diagramas. En la Fig. 17, mostramos en detalle la actividad de creación de *Mockups* y *WebSpec*; comenzamos creando *mockups* para darles un contexto a los clientes. Luego, creamos los diagramas *WebSpec* de acuerdo a los requerimientos de éstos y, para chequear el comportamiento esperado, simulamos los diferentes caminos de interacción. Una vez que hayamos acordado el requerimiento, la actividad de creación de *Mockups* y *WebSpec* termina.

Para poder dar soporte a la simulación de la aplicación, *WebSpec* permite la asociación entre las interacciones con los *mockups* y entre los *widgets* de *WebSpec* con sus correspondientes elementos de UI en el *mockup*. Por medio de esta asociación, podemos cambiar entre la especificación de *WebSpec* con el ejemplo de UI que tenemos en el *mockup*, lo que ayuda a entender el requerimiento. Los *mockups* pueden ser creados con herramientas como *Balsamiq* [BAL12], *Axure* [AXW12] o HTML plano. Por ejemplo, en la Fig. 18, mostramos un *mockup* para la página encargada de agregar alertas con *Balsamiq*. El *mockup* muestra la información que debe aparecer en la página: el nombre de las alertas, su descripción, su tipo y los links a la página de inicio y cerrar sesión.

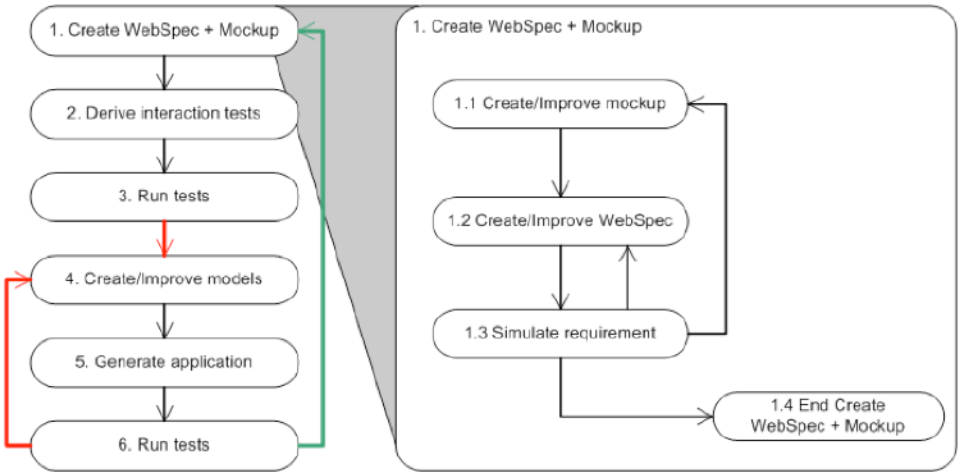


Fig. 17. La simulación de *WebSpec* en el contexto de *WebTDD*

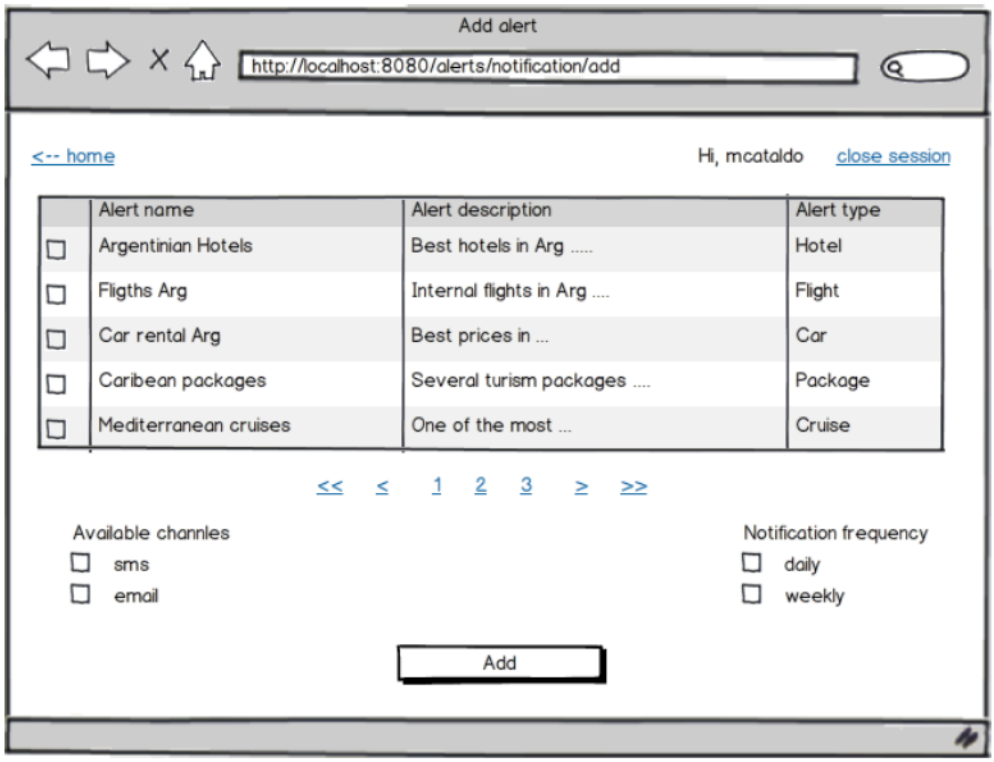


Fig. 18. *Mockup* de la interacción *AddAlert* creado con *Balsamiq*

3.2.3 VALIDACIÓN DE REQUERIMIENTOS

Los requerimientos nuevos deben ser validados para garantizar su correcta implementación, mientras que los previos deben seguir funcionando como se espera. Sin embargo, realizar esta tarea eficientemente es complicado, y por lo tanto mantener los requerimientos actualizados se vuelve extremadamente importante.

Una manera conocida de validar los requerimientos consiste en ejecutar tests automatizados (que expresan los requerimientos) sobre la aplicación. Si uno de estos tests falla, entonces un requerimiento no es satisfecho por la aplicación. En particular, los tests de interacción juegan un papel fundamental en la industria, ya que ejecutan un conjunto de acciones de la misma forma que un usuario lo haría en un navegador WEB, y por lo tanto su uso continúa creciendo [MW03]. En la Fig. 19, mostramos con más detalle las actividades ejecutadas durante el ciclo *WebTDD*; primero, necesitamos elegir un conjunto de diagramas *WebSpec* que expresen el nuevo requerimiento (Paso 2.1) y, de ellos, derivar en forma automática un conjunto de tests de interacción (Paso 2.2). Luego, ejecutamos dichos tests (*Test Suite*) (Paso 3.1), los cuales han sido derivados utilizando algún *framework* de tests (por ejemplo *JUnit*) (Paso 3.2).

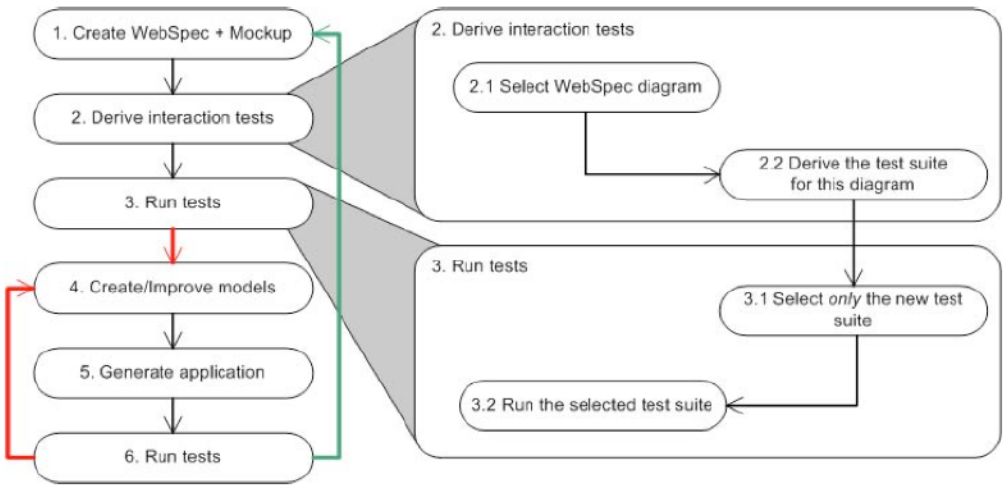


Fig. 19. Derivación de tests de *WebSpec* en el contexto de *WebTDD*

El conjunto de tests se construye a partir de un diagrama de *WebSpec* mediante la creación de un test para cada escenario que la aplicación debe satisfacer. Para capturar los conceptos básicos de los tests, se ha creado un metamodelo (Fig. 20), que es independiente de la tecnología de test utilizada. El metamodelo contiene las clases *Test* y *TestSuite,* que conceptualizan un test y un conjunto de tests. Un test tiene una secuencia de acciones: las afirmaciones sobre objetos de la interfaz o las acciones realizadas por el usuario a través de la aplicación. Ambos casos están cubiertos por la jerarquía *TestItem*.

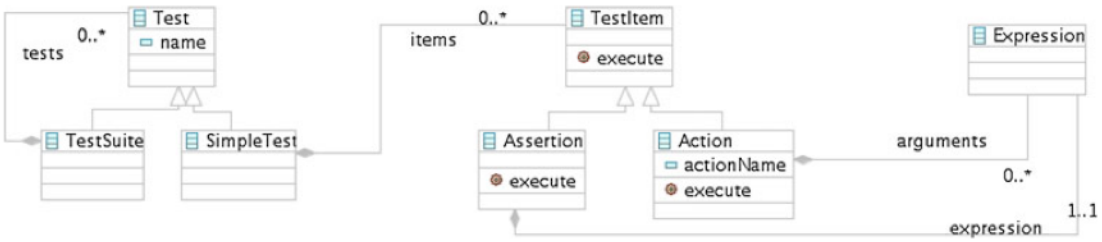


Fig. 20. Metamodelo de test

Para construir el conjunto de test, transformamos cada escenario en un *SimpleTest* (Fig. 20) mediante la ejecución de la siguiente versión simplificada del algoritmo. Al igual que en las simulaciones, utilizamos generadores para generar datos de acuerdo con la especificación cuando una expresión hace referencia a él. El conjunto de tests (*TestSuite*) se obtiene por simple composición (véase la relación de composición en el metamodelo de la Fig. 20) de los casos de *SimpleTest* anteriores.

(1) Crear un test T para cada escenario C
(2) Agregar un ítem para abrir la URL de la aplicación en T
(3) Para cada elemento E en el escenario C {
(4) si (E es una interacción) {
(5) Agregar una aserción acorde con la invariante de E

```
(6)         } else {
(7)             Para cada acción A en la transición E {
(8)                 Agregar una ejecución de la acción A
(9)             }
(10)        }
(11) }
```

El algoritmo funciona de la siguiente manera: la línea 1 crea el modelo de test, y la línea 2 genera la acción para abrir la aplicación. Para cada elemento del camino, si se trata de una interacción (4), afirmamos su invariante (5), y si se trata de una transición (7), llevamos a cabo las acciones que nos permiten navegar de una interacción a otra (7 - 9).

Después de que se crea una instancia del metamodelo de test, la aplicación puede ser validada utilizando un *framework* de test dependiente de la tecnología utilizada, que opera sobre el navegador WEB. En la Sección. 3.3 se proporcionan más detalles sobre la ejecución de la derivación de test en el *plugin* de *Eclipse* de *WebSpec*.

Como se mencionó anteriormente, las aplicaciones WEB tienden a cambiar muy rápido, por lo que el registro de los cambios de los requerimientos que se van produciendo es importante para mejorar el proceso de desarrollo. En la siguiente subsección, se muestra cómo los cambios de requerimientos son capturados (Sección 3.2.4) y utilizados más adelante para facilitar la evolución de la aplicación en desarrollo.

3.2.4 CAMBIOS EN LOS REQUERIMIENTOS

La captura de los cambios en los requerimientos es una característica importante para predecir el impacto de estos en la aplicación. Aunque algunos artefactos para la captura de requerimientos proveen extensiones para soportar

el control de cambios, en el campo de la ingeniería WEB estos aspectos han sido ignorados.

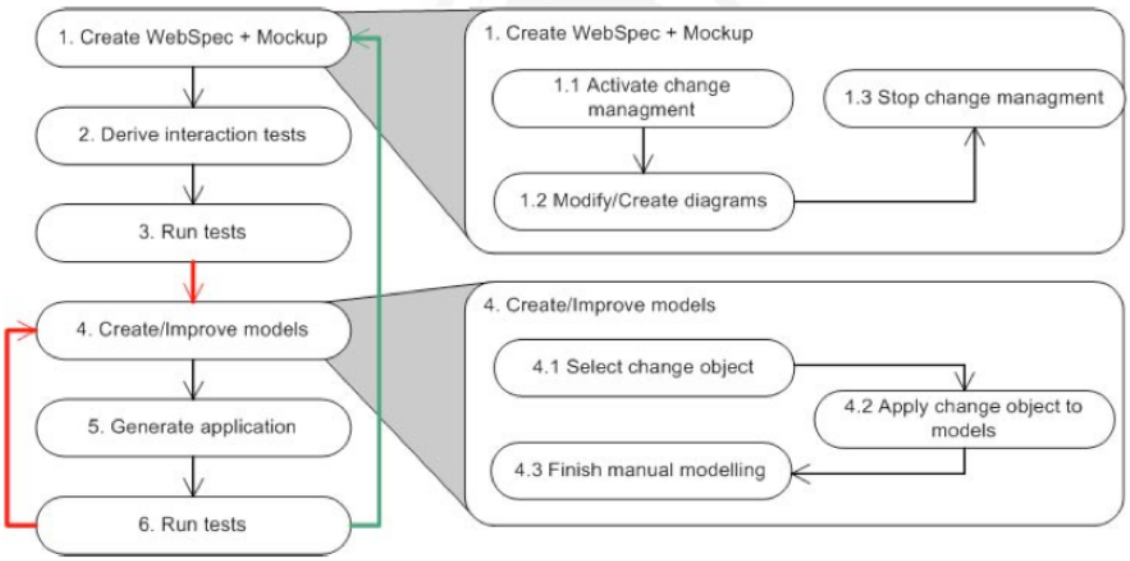


Fig. 21. Derivación de la aplicación en el contexto de *WebTDD*

En *WebSpec*, los cambios son grabados en objetos de cambio (*change objects*) que agrupan un conjunto de cambios en los diagramas. Los *change objects* son creados aún en las fases iniciales (cuando el diagrama está siendo creado). Los diagramas *WebSpec* pueden tener diferentes cambios de grano grueso, como el agregado o borrado de una interacción o transición. Estos elementos también pueden ser modificados por el agregado o borrado de *widgets* a una interacción, cambio en los invariantes, etc. Respecto a las transiciones, podemos agregar o modificar sus precondiciones, cambiar su origen y destino, o las acciones que las activan. Todos estos posibles cambios son presentados en el metamodelo de la Fig. 22. Cuando un usuario modifica un diagrama, un objeto de cambio es creado y la secuencia de cambios es grabada como instancia de este metamodelo.

Fig. 22. Metamodelo para los objetos de cambio

En la Fig. 21, se muestran las actividades en el contexto de *WebTDD*; cuando estamos creando o modificando diagramas, activamos el control de cambios de *WebSpec* para grabar dichos cambios. Luego, cuando comenzamos con las tareas de modelado, podemos aplicar estos cambios en forma semiautomática a nuestros modelos para mejorarlos. Como *WebSpec* no soporta todos los tipos de cambios (especialmente aquellos relacionados con cómo fue modelada la aplicación), debemos continuar con las tareas de modelado en forma manual.

Como ejemplo, supongamos que agregamos una interacción de registro (*Register*) con sus *widgets* y un link desde la interacción *Login* (Fig. 23). Este cambio en el diagrama genera un nuevo *change object* que contiene los siguientes elementos: una nueva interacción (*Register*), una nueva navegación (*Login* → *Register*), un nuevo link (*register*) en la interacción *Login* y un nuevo conjunto de *widgets* en la interacción *Register*.

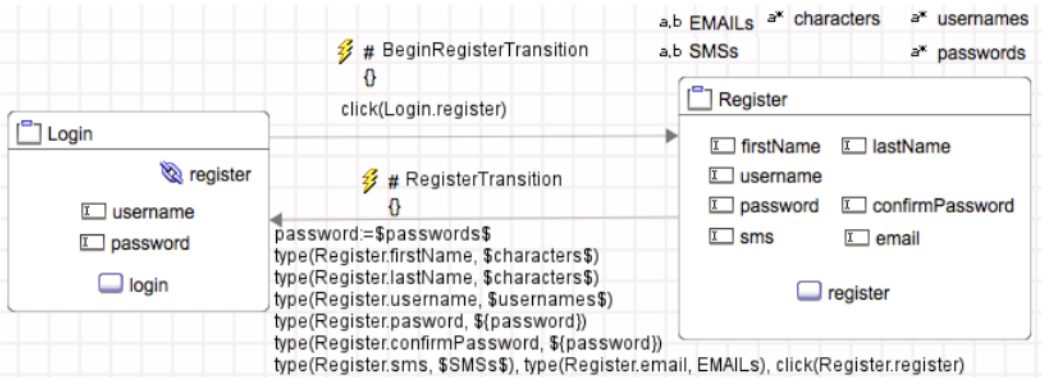


Fig. 23. Diagrama con la interacción *Register*

El manejo de cambios de requerimientos sirve para muchos propósitos útiles; sin embargo, se hará hincapié en cómo actualizar la aplicación de manera semiautomática dados los cambios en los requerimientos. Dado que los *change objects* representan cambios a nivel de *WebSpec* (requerimientos), se desacopla el proceso de actualización de la aplicación al proveer diferentes manejadores de cambios. Un manejador de cambio es un componente responsable de mapear los cambios en los diagramas a una tecnología concreta y guardar las asociaciones entre los elementos de *WebSpec* y los de las tecnologías.

Para mantener la discusión en un nivel conceptual y mostrar un ejemplo concreto, se asume que la aplicación en desarrollo está diseñada con clases que ya tienen una versión de la aplicación. En la Fig. 24, se muestra un diagrama de clases que contiene el modelo de UI de la aplicación antes de la aplicación de los cambios de la Fig. 23.

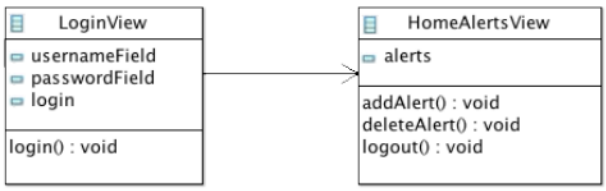


Fig. 24. Diagrama de clases

Para actualizar la aplicación después de haber aplicado los cambios, se tiene que definir una serie de mapeos entre los cambios de *WebSpec* y las implementaciones concretas. En un diseño basado en clases, se definen los siguientes mapeos:

- Nueva interacción → Una nueva clase es creada.
- Nuevo *widget* → Una nueva variable de instancia es agregada, así como también su método de inicialización.

- Actualización del nombre de una interacción o *widget* → La clase o la variable de instancia son renombradas.
- Borrado de una interacción → La clase se borra si ninguna otra la referencia.
- Borrado de un *widget* → La variable de instancia es borrada junto con su método de inicialización si la variable no tiene referencia alguna.

Si usamos los mapeos previamente detallados, podemos actualizar el modelo de UI automáticamente y obtener un nuevo modelo de UI (Fig. 25). La clase *RegisterView* es creada con sus correspondientes variables de instancia. También, la clase *LoginView* es modificada con una nueva variable de instancia *register* que contiene un link a *RegisterView*.

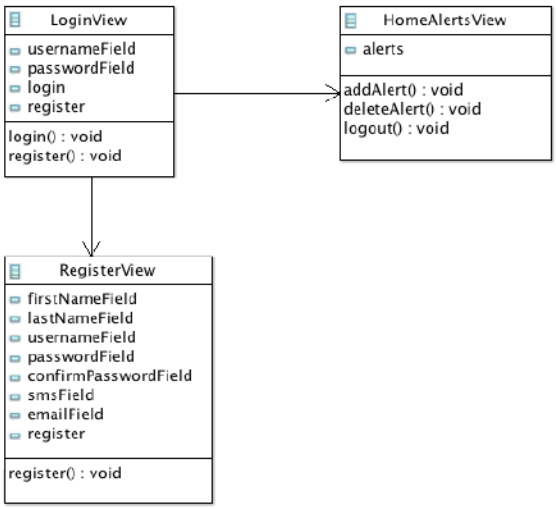


Fig. 25. Nuevo diagrama de clases

3.3 SOPORTE DE *WEBSPEC*

Una herramienta para *WebSpec* ha sido implementada como *plugin* de *Eclipse* por medio de tecnologías tales como EMF [EMF12] y GMF [GMF12] y está disponible actualmente como proyecto *open source*.

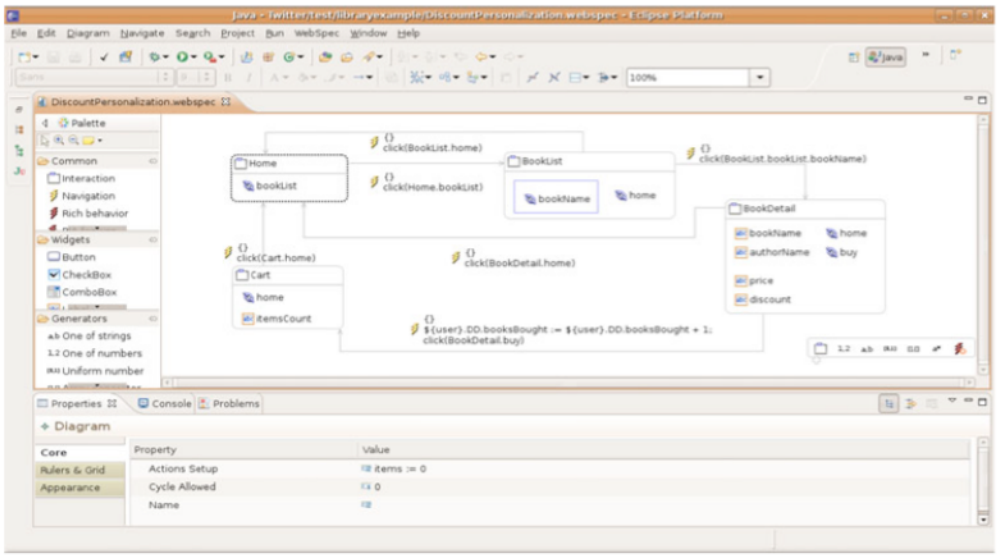


Fig. 26. *Plugin* de *Eclipse* para *WebSpec*

El *plugin* soporta las siguientes características:

- Creación de diagramas *WebSpec*: un editor visual permite la creación, modificación y actualización de los diagramas. Las propiedades de los elementos pueden ser modificadas si se selecciona cada elemento y se actualizan los editores de propiedades en la vista de propiedades.
- Asociación con *mockups* HTML: si tomamos como ventaja el *framework* de Eclipse, los *mockups* HTML son archivos dentro del proyecto. El editor permite seleccionar una interacción y su *mockup* HTML fácilmente. La asociación entre los *widgets* es realizada editando la propiedad *location* del *widget WebSpec* en la vista de propiedades.
- Simulación de la aplicación: al utilizar la asociación previa, el *plugin* abre los *mockups* en un navegador WEB y muestra descripciones de cuál sería

el comportamiento esperado. Esta característica ha sido implementada mediante la extensión del mecanismo de comunicación de *Selenium* [SEL12] y la utilización de un *plugin* de *JQuery* [JQY12] para mostrar las descripciones.

- Derivación de tests a *Selenium*: Como mostramos anteriormente, cada diagrama es transformado en un modelo de tests. Luego el *plugin* permite la traducción del modelo de tests a tests de *Selenium.*
- Manejo de cambios: A través del uso del patrón *observer* [GHJV95] de EMF, nos registramos para recibir notificaciones de los cambios en el diagrama, y, así, el *plugin* crea el modelo de cambios. El usuario del *plugin* es quien decide si comenzar a grabar los cambios o no. Cuando algunos cambios han sido capturados, y el usuario detiene el grabado, el modelo de cambios es grabado en un archivo para ser usado posteriormente.
- Generación o actualización de clases GWT y *Seaside*: Al utilizar el modelo de cambios grabado con anterioridad, el modelo de UI puede ser generado de forma automática. Actualmente, el *plugin* soporta la generación de clases GWT y *Seaside* y maneja no solo la primera versión de los cambios, sino también los cambios incrementales.

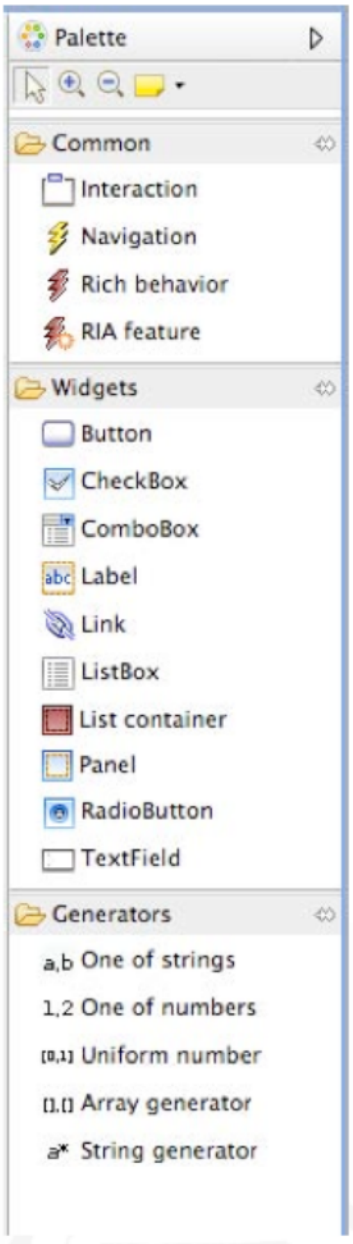


Fig. 27. La paleta de *WebSpec*

La Fig. 26 muestra una pantalla del *plugin* de *Eclipse*. En la Fig. 27, podemos ver más detalles de la paleta de *WebSpec* que permite la creación de cada elemento *WebSpec* realizando un *drag and drop* de cada elemento sobre el diagrama. Luego, si seleccionamos un elemento, podemos editar sus propiedades en la vista de propiedades de *Eclipse* (Fig. 28). En las próximas

subsecciones daremos más detalles de cómo se han implementado cada una de las características presentadas por el *plugin*.

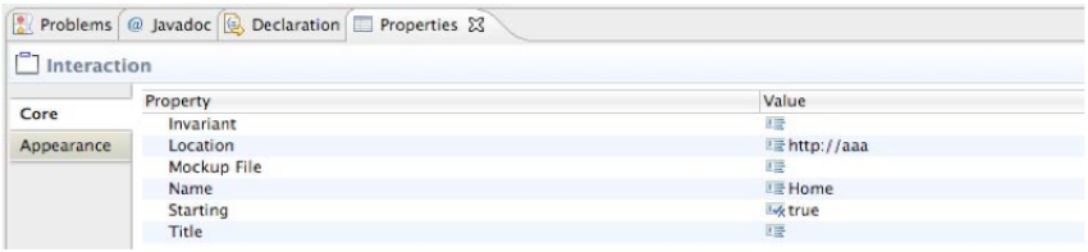


Fig. 28. Las propiedades de un diagrama *WebSpec*

3.3.1 SIMULACIÓN

La característica de simulación implica la implementación de tres elementos: la transformación entre *WebSpec* y los modelos de simulación, la asociación entre los *mockups* y la ejecución de la simulación. La transformación entre *WebSpec* y los modelos de simulación ha sido implementada directamente en Java debido a que es mucho más sencillo realizar los algoritmos de cómputo de caminos en este lenguaje que usar QVT [QVT12]. Para realizar la transformación, simplemente abrimos el menú de *WebSpec* (Fig. 29) y elegimos el ítem *Simulate*.

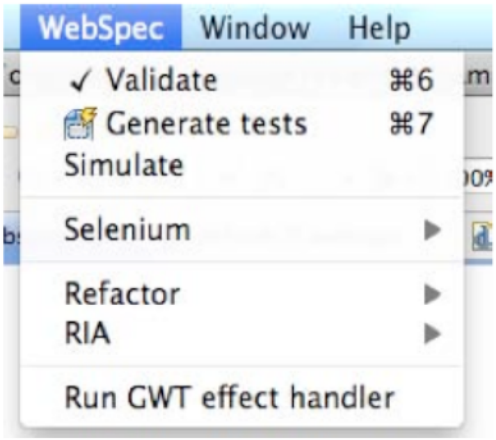


Fig. 29. El menú de *WebSpec*

La asociación con los *mockups* ha sido implementada fácilmente por medio de las herramientas del ambiente de *Eclipse*. Agregamos una nueva propiedad para las interacciones y los *widgets* que abre una ventana de diálogo en la cual se puede elegir un archivo que, a su vez, permite elegir el *mockup* HTML.

La simulación en sí fue la parte más compleja de implementar y requirió la extensión del *framework Selenium*. Se utilizó el mecanismo de comunicación de *Selenium* para abrir el navegador WEB y ejecutar las acciones en él. En la Fig. 30, se pueden ver descripciones sobre los *mockups* con un *plugin* de *JQuery*. Para hacerlo funcionar, se tuvo que extender el *framework* de *Selenium* para que cargue dichas librerías y muestre las descripciones cuando sea necesario. Debemos notar que el mismo *mockup* (el cual puede ser más rico que una interacción debido a que posee más *widgets*) puede ser utilizado en múltiples y diferentes simulaciones. Este acercamiento mantiene el *mockup* en el mismo estado en el cual fue construido sin quitar ninguno de los *widgets* existentes, ya que confundirían a los clientes acerca de la presencia o ausencia de aquellos.

Fig. 30. Ejemplo de simulación en *WebSpec*

3.3.2 VALIDACIÓN DE REQUERIMIENTOS

Para permitir la validación de requerimientos, se realizaron dos tareas: la transformación de los diagramas en modelos de tests y, luego, la derivación de esos tests a una tecnología en particular. La transformación entre los modelos ha sido implementada con la arquitectura y las transformaciones ya existentes para la simulación, debido a que ambas utilizan algoritmos de cómputo de caminos.

Con el fin de transformar los modelos de tests a unos dependientes de la tecnología, se utilizaron transformaciones de modelos a texto. Actualmente, el *plugin* soporta la derivación de tests a *Selenium* y *Webdriver* [WDR12]. Como ejemplo, se muestra a continuación el código generado para nuestro caso de derivación en el *framework Selenium*:

```
(01) selenium.open("http://localhost:8080/login.html");
(02) selenium.type("id=login.usernameField", "mcataldo");
(03) selenium.type("id=login.passwordField", "123456");
(04) selenium.click("id=login.login");
(05) selenium.waitForPageToLoad("30000");
(06) assertTrue(selenium.getText("id=halerts.username").equals("mcataldo"));
(07) selenium.click("id=halerts.logout");
```

La línea 01 abre la aplicación en el navegador WEB. Las líneas 02 y 03 completan los campos de nombre de usuario y contraseña. La línea 4 hace *click* en el botón *login*. La línea 05 espera que la página *home* de alertas sea cargada

por el navegador. La línea 06 asegura que el usuario que ingresó sea "mcataldo". Y, finalmente, la línea 07 sale de la aplicación.

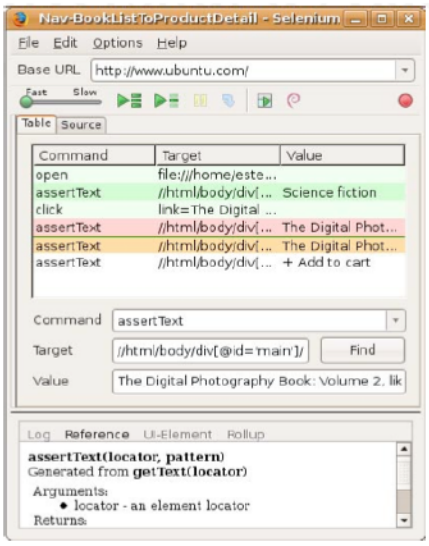


Fig. 31. Un test que falla

Como ejemplo, los tests de *Selenium* pueden ser ejecutados en el *Selenium* IDE; la Fig. 31 muestra un test que falla e indica que un requisito aún no ha sido correctamente implementado en la aplicación.

3.3.3 CAMBIOS EN LOS REQUERIMIENTOS

Cuando un diagrama es modificado, se graban sus cambios en un archivo de cambios. Este archivo no es más que una serialización del modelo de cambios en formato XML. Para capturar los cambios, utilizamos el patrón *observer* e, incrementalmente, vamos construyendo el modelo de cambios; luego lo serializamos a XML. Los archivos de cambios son leídos y utilizados para actualizar la aplicación mediante manejadores de cambios (un componente que se encarga de mapear los cambios realizados en *WebSpec* a una

tecnología en particular). El *plugin* soporta la generación de clases y métodos compatibles con *Seaside* y GWT.

Como ejemplo del uso de manejadores de cambios, mostramos a continuación el uso de objetos de cambio en nuestro ejemplo de actualización (el agregado de la registración que hemos mostrado previamente) en GWT. Con el fin de mantener la discusión acotada, mostramos la clase *RegisterView* creada por el manejador de cambios de GWT.

Básicamente, las líneas 09-15 definen las variables de instancia que representan a los *widgets,* y las líneas 21-29 inicializan estos objetos con sus respectivas clases GWT. Debemos notar que la clase *RegisterView* se extiende desde *VerticalPanel* (una clase base en GWT para la implementación de interfaces gráficas de usuario).

```
(01)    package org.webspeclanguage.re;
(02)
(03)    import com.google.gwt.user.client.ui.VerticalPanel;
(04)    import com.google.gwt.user.client.ui.TextBox;
(05)    import com.google.gwt.user.client.ui.Button;
(06)
(07)    public class RegisterView extends VerticalPanel {
(08)
(09)    private TextBox firstName;
(10)    private TextBox lastName;
(11)    private TextBox username;
(12)    private TextBox password;
(13)    private TextBox confirmPassword;
(14)    private TextBox sms;
(15)    private TextBox email;
(16)    private Button register;
(17)
(18)    public RegisterView() {
(19)        this.initializeComponent();
(20)    }
(21)
(22)    public void initializeComponent() {
(23)        this.firstName = new TextBox();
(24)        this.lastName = new TextBox();
(25)        this.username = new TextBox();
(26)        this.password = new TextBox();
(27)        this.confirmPassword = new TextBox();
```

```
(28)                this.sms = new TextBox();
(39)                this.email = new TextBox();
(30)                this.register = new Button();
(31)        }
(32)  }
```

CAPÍTULO 4

RESOLUCIÓN CONCEPTUAL

El objetivo de este capítulo es contar cómo se llega a una alternativa de solución conceptual para poder mejorar el entendimiento de los diagramas de *WebSpec*. Para ello, se hace una breve mención de la problemática de *WebSpec* con respecto a la compresión de los diagramas (Sección 4.1); luego, se enumera cómo distintas alternativas manejan las transformaciones o derivaciones de un modelo a otro (Sección 4.2); posteriormente, analizamos el modelo de *WebSpec* (Sección 4.3); y, finalmente, presentamos un algoritmo que da soporte a una herramienta que intenta aminorar la problemática de *WebSpec* (Sección 4.1).

4.1 COMPRENSIÓN DE UN DIAGRAMA DE *WEBSPEC*

Como se ha explicado en el capítulo introductorio (Sección 1.4), esta tesis está motivada por la necesidad de ofrecer una herramienta para una mejor compresión de la utilización de *WebSpec*. Particularmente, la necesidad se presenta en aquellos usuarios que comienzan a utilizar *WebSpec* por primera vez. Muchas veces, los analistas funcionales tienen que interactuar directamente con los usuarios o clientes para poder recolectar y especificar los requerimientos. La idea es que el mismo usuario especifique los requerimientos, dado que él es quien conoce todos los pormenores de los aspectos funcionales

que se están relevando. De esta manera, hay que suministrarle al usuario una herramienta (*WebSpec*) con la cual pueda especificar los requerimientos, y que ésta sea lo suficientemente amigable como para que el usuario no tenga inconvenientes en su utilización. Es en este último aspecto donde se encuentra una falencia en *WebSpec*: los diagramas requieren un cierto conocimiento formal para poder ser utilizados correctamente. Con el objetivo de mejorar la etapa de obtención (particularmente la creación de diagramas, su interpretación y comprensión), los diagramas de *WebSpec* se podrán derivar o transformar en un formato textual estructurado. Este formato de texto estructurado que se obtiene como resultado es importante para reducir la diferencia en el entendimiento de los diagramas. Y es de esta forma en la que los usuarios que utilizan *WebSpec* por primera vez cuentan con una herramienta que les ofrece una traducción de los diagramas a un formato más informal y cercano al lenguaje que están acostumbrados a manejar.

4.2 DISTINTAS ALTERNATIVAS DE TRANSFORMACIÓN O DERIVACIÓN

En el Capítulo 2, se hizo un pequeño análisis del estado del arte de algunas metodologías de desarrollo; puntualmente se hizo hincapié en la ingeniería de requerimientos (obtención, especificación y validación de requerimientos). Al final del Capítulo (Sección 2.4 y 2.5), se hicieron comparaciones y conclusiones acordes al análisis realizado. A partir de los resultados de ese capítulo, analizaremos cuáles metodologías cuentan con algún mecanismo de transformación o derivación y de qué manera son realizados.

En la Fig. 4, se puede observar que las metodologías que ofrecen algún tipo de transformación o derivación son *OOWS*, *NDT* y *UWE*. *OOWS* provee una generación automática de los modelos de navegación desde las descripciones

de las tareas por medio de reglas de transformación de grafos. *NDT* define un metamodelo y permite la transformación de requerimientos a modelos conceptuales (modelos de contenido y navegación) mediante el uso de reglas *QVT* [QVT12]. Por último, *UWE* es una de la metodologías que más alternativas provee a la hora de realizar transformaciones; éstas van desde requerimientos hasta modelos de contenido, de requerimientos a modelos de arquitectura, de requerimientos a modelos de navegación, etc. Estas transformaciones utilizan distintas estrategias para poder llevar a cabo las transformaciones. Algunas de estas estrategias son: *QVT*, *ATL* [JK05] y transformaciones de grafos.

***QVT* (*Queries, Views and Transformations*)**

QVT es el estándar del OMG para escribir transformaciones. La especificación de *QVT* tiene una naturaleza híbrida, relacional (o declarativa) y operacional (o imperativa). Comprende tres lenguajes M2M diferentes (model *to model*): dos lenguajes declarativos llamados *Relations* y *Core*, y un tercer lenguaje, de naturaleza imperativa, llamado *Operational Mappings*. Esta naturaleza híbrida fue introducida para cubrir diferentes tipos de usuarios con diferentes necesidades, requisitos y hábitos. La especificación de QVT define tres paquetes principales, uno por cada lenguaje definido: *QVTCore*, *QVTRelation* y *QVTOperational*.

ATL

ATL (*Atlas Transformation Language*) [ATL] consiste en un lenguaje de transformación de modelos y un *toolkit* creados por ATLAS Group (INRIA y LINA) que forma parte del proyecto GMT de *Eclipse*. En él se presenta un lenguaje híbrido de transformaciones declarativas y operacionales de modelos que, si bien es del estilo de *QVT*, no se ajusta a las especificaciones. Aunque la sintaxis de *ATL* es muy similar a la de *QVT*, no es compatible con este último. Tiene herramientas que realizan una compilación del código fuente a *bytecodes* para una máquina virtual que implementa comandos específicos para la ejecución de transformaciones. *ATL* forma parte del *framework* de gestión de

modelos AMMA que se encuentra integrado en *Eclipse* y EMF. *ATL* posee un algoritmo de ejecución preciso y determinista. El *toolkit* es de código abierto.

Transformación de grafos

Esta categoría reúne las propuestas de transformación de modelos basadas en la teoría de transformaciones de grafos [EEKR99]. En particular, este tipo de transformaciones actúa sobre grafos etiquetados y con atributos, que pueden pensarse como representaciones formales de modelos de clase simplificados. Las principales propuestas en esta categoría incluyen a *AGG* [TAG03], *AToM3* [JLHV02], *VIATRA* [VVVP02, DVAP04], *GReAT* [AKS03], *UMLX* [EDW03], *MOLA* [KBC04], y *Fujaba* [FTS12]. *AGG* y *AToM3* son sistemas que implementan directamente la propuesta teórica para grafos con atributos y también las transformaciones sobre estos grafos. Las reglas de transformación son unidireccionales e *in-place*. Cada regla de transformación de grafo consta de dos patrones, el derecho y el izquierdo. La aplicación de una regla implica localizar un patrón izquierdo en el modelo y reemplazarlo por el correspondiente patrón derecho.

4.3 EL MODELO INTERNO DE *WEBSPEC*

WebSpec está formalmente definido en el metamodelo que se muestra en la Fig. 32. Un diagrama (instancia de la clase *Diagram*) consta de interacciones (instancias de la clase *Interaction*) y transiciones ya sea de navegación o comportamiento rico (instancias de la clase *Transition*). Una instancia de la clase *Interaction* conoce su nombre, sus transiciones y su maqueta de UI asociada (*mockup*). Una instancia de *Transition* conoce su origen, destino, sus precondiciones, y la secuencia de acciones que hace posible la transición. Por último, una instancia de la clase *Interaction* conoce su contenedor de *widgets*, que puede contener muchas instancias de *AbstractWidget* (*Widget* o *Container*).

Cada *widget* también puede estar asociado con su maqueta de UI mediante su atributo de ubicación.

Claramente, se puede ver que un diagrama de *WebSpec* es un grafo dirigido: las interacciones representan los nodos, y las navegaciones, las aristas.

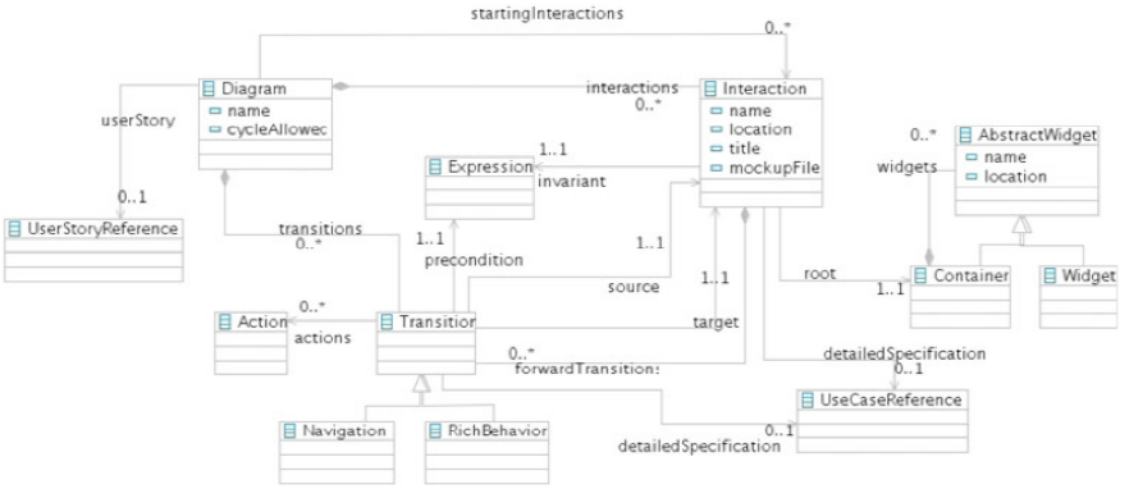


Fig. 32. Metamodelo de *WebSpec*

4.4 SOLUCIÓN CONCEPTUAL

En esta sección, se presenta la solución conceptual a la problemática mencionada en las Secciones 1.4 y 4.1. En la Sección 1.5, se presentó una propuesta para abordar los temas comentados en las Secciones 1.4 y 4.1. A continuación, veremos cómo se llega a definir la propuesta de la Sección 1.5.

El hecho de que los usuarios presenten algún problema de comprensión sobre los diagramas de *WebSpec* hizo que se piense en un tipo de traducción de los diagramas. Los diagramas de *WebSpec* representan un dominio. Éste tiene una semántica; además, está escrito con una sintaxis formal que no es fácil de entender. Por esta razón, se decidió hacer una traducción de los diagramas, por

la cual poder llevarlos a una representación que cualquier persona podría llegar a interpretar con un mínimo contexto del tema.

La implementación que se realiza en esta tesis proporciona dos alternativas para estructurar el documento de traducción. Una de ellas es un formato tabular con tablas, y la otra es un formato enumerativo que utiliza secciones. Los usuarios están bastante familiarizados con las tablas, planillas de celdas y con la organización por medio de secciones. Por otro lado, generalmente los documentos están organizados por medio de índices, y lo que especifican los índices son secciones.

Dado un diagrama de *WebSpec* en un instante determinado, el cual contiene uno o varios escenarios de la aplicación, la pregunta principal es: ¿Cómo se llega de los diagramas a los documentos? Para poder responder esta pregunta, se tiene que elaborar un algoritmo que permita obtener un documento dado un diagrama de *WebSpec*. A continuación, se especifica el algoritmo en pseudocódigo; éste basa su estrategia de solución en dos partes: una es la forma en la que los diagramas de *WebSpec* son internamente modelados (Sección 4.3), y la otra especifica una relación directa entre cada uno de los elementos de los diagramas (interacciones y navegaciones). Como se mencionó en la Sección 4.3, dado que los diagramas tienen la naturaleza de grafos, podemos obtener cada uno de los posibles caminos del grafo. Por lo tanto, dado un diagrama (grafo), se puede obtener cada uno de sus escenarios (caminos). Por otro lado, una vez obtenidos cada uno de esos escenarios (caminos) del diagrama (grafo), se puede ir recorriendo los escenarios e ir traduciendo cada uno de los elementos (interacción o navegación) que se van visitando. Para esto, se necesita tener un diccionario en el cual se puedan establecer las traducciones. Se verá que dada una clave en el diccionario ésta se puede traducir hasta en dos posibles valores dependiendo del tipo de clave. En la Fig. 33 se especifica este tipo de asociaciones.

```
00    Crear el documento
01    Obtener los escenarios del diagrama
02    Mientras haya escenarios {
03        Agregar el inicio del escenario al documento
04        Mientras no esté en el final del escenario {
05            Obtener traducción del elemento
06            Agregar la traducción al documento
07        }
08        Agregar el final de escenario al documento
09    }
10    Guardar el documento
```

Del pseudocódigo presentado, se puede deducir lo siguiente:

- Dada la intención de traducir el diagrama, en la línea 00 se crea un documento que puede ser un archivo de WORD o HTML.
- En la línea 01, se obtienen los escenarios del diagrama que se quiere traducir. Esta obtención se realiza por medio de un algoritmo de búsqueda en profundidad DFS (*Depth First Search*).
- En la línea 02, se recorre la lista de los escenarios del diagrama con un iterador.
- En la línea 03, se agrega el encabezado del escenario al documento. Tradicionalmente, esto puede ser un nombre, una descripción y hasta una imagen del escenario.
- En la línea 04, se recorre cada uno de los elementos del escenario (interacciones y navegaciones) por medio de un iterador.
- En la línea 05, se obtiene la traducción del elemento que se esta visitando durante la recorrida del escenario.
- En la línea 06, se agrega al documento la traducción obtenida en el paso anterior. Dependiendo de la estrategia que se haya elegido (documento

WORD tabular, WORD enumerado o HTML tabular), se da formato a la traducción obtenida en el paso anterior.

- En la línea 08, se agrega al documento el pie de página del escenario.
- En la línea 10, se guarda el documento en una memoria no temporal.

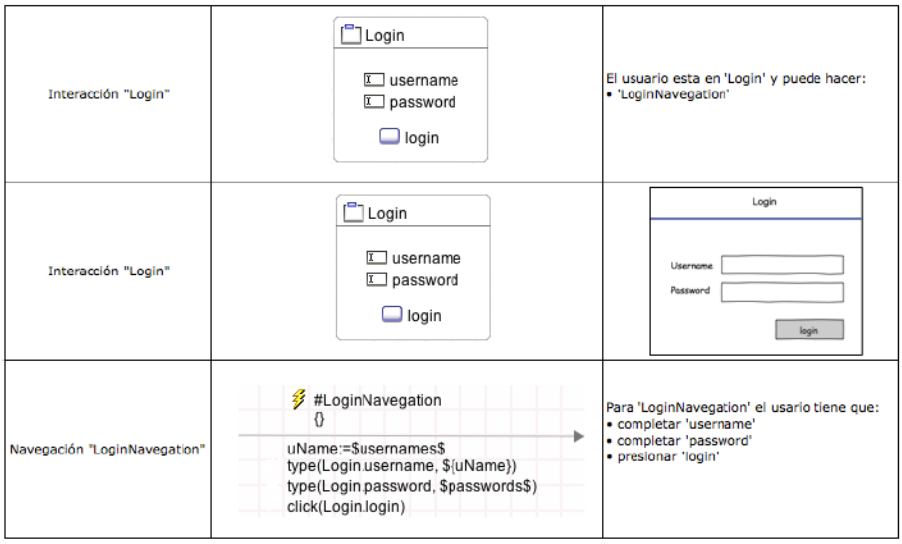


Fig. 33. Diccionario

Un punto muy importante es el que se desarrolla en la línea 06 del algoritmo. Dependiendo de cuál haya sido la estrategia con la cual se va a conformar la estructura del documento, es como se da formato a cada uno de los elementos (interacciones y navegaciones) que se van recorriendo en el escenario.

En la Fig. 33, se puede observar el diccionario utilizado para realizar las traducciones. Cuando se recorre un escenario y se encuentra una interacción o navegación, éstas son traducidas de dos maneras distintas. Las interacciones pasan por un proceso doble: se traduce la interacción a un texto que explica tanto a esa interacción como a las posibilidades de navegación, y por otro lado se obtiene una imagen, si es que la interacción tiene asociada un *mockup*. Por otro lado, las navegaciones son traducidas a texto; de esta manera, se nombra

la navegación, se enumeran las precondiciones y se enumeran las acciones que contienen.

Dependiendo de la estrategia que se haya optado para estructurar el documento, cada interacción va a mostrar una imagen de sí misma (como aparece en el diagrama), una descripción textual y una imagen, si es que tiene un *mockup* asociado. Por su parte, las navegaciones van a mostrar una imagen tal cual se representa en el diagrama y una descripción textual.

Los documentos con estructura enumerativa serán organizados con una lista de cada uno de los escenarios del diagrama. Cada uno de estos escenarios será presentado así:

1. El camino que define al escenario
2. Una imagen del diagrama completo
3. Una sección con el nombre de la interacción o navegación
4. Una subsección con la imagen de interacción o navegación
5. Otra subsección con la explicación de la interacción o navegación
6. Otra subsección con una imagen del *mockup,* si es que es una interacción y tiene un *mockup*
7. Se repiten los pasos del punto 3 al 6 hasta que se hayan recorrido todas las interacciones y navegaciones del escenario
8. Se repiten los pasos del punto 1 al 7 hasta que se hayan recorrido todos los escenarios del diagrama

Por otro lado, los documentos con estructura tabular también serán organizados con una lista de cada uno de los escenarios del diagrama. Cada uno de estos escenarios será presentado así:

1. El camino que define al escenario
2. Una imagen del diagrama completo
3. Una tabla con filas que explican interacciones o navegaciones

4. Cada una de estas filas tiene una primera columna con la imagen de la interacción o navegación
5. Cada una de estas filas tiene una segunda columna con la explicación de la interacción o navegación
6. Cada una de estas filas tiene una tercera columna que puede tener un *mockup,* si es que es una interacción y además contiene un *mockup*
7. Para cada escenario del diagrama, se irá completando el documento de acuerdo a los pasos que van desde el 1 al 6.

CAPÍTULO 5

IMPLEMENTACIÓN

Con el objetivo de llevar a la práctica el mecanismo de generación de documentos propuesto en el capítulo anterior, se llevó a cabo la implementación de un módulo de generación de escenarios a partir de diagramas *WebSpec*. El objetivo de este capítulo es presentar los detalles de la implementación del módulo en cuestión.

La organización del presente capítulo comienza con una sección que presenta una breve descripción de los patrones de diseño aplicados (Sección 5.1); a continuación, una sección donde se muestra el diseño y cómo se llevó a cabo la implementación del módulo de generación (Sección 5.2); y, finalmente, las tecnologías utilizadas (Sección 5.3).

5.1 PATRONES DE DISEÑO

En esta sección, se presentan los patrones de diseño que se aplicaron en la implementación. A continuación, se presenta una breve mención acerca de ellos. Una completa descripción de los patrones de diseño se puede encontrar en [GHJV95] y [MFPEA].

Builder

Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones. Uno de los motivos por los cuales se utiliza es porque los constructores y *factories* estáticos no escalan muy bien con un número grande de parámetros.

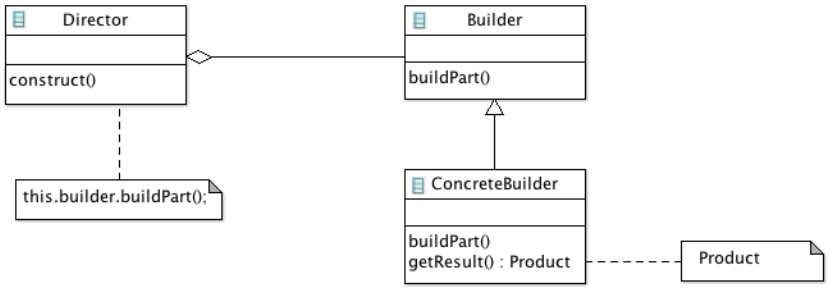


Fig. 34. Patrón de diseño *Builder*

Visitor

Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

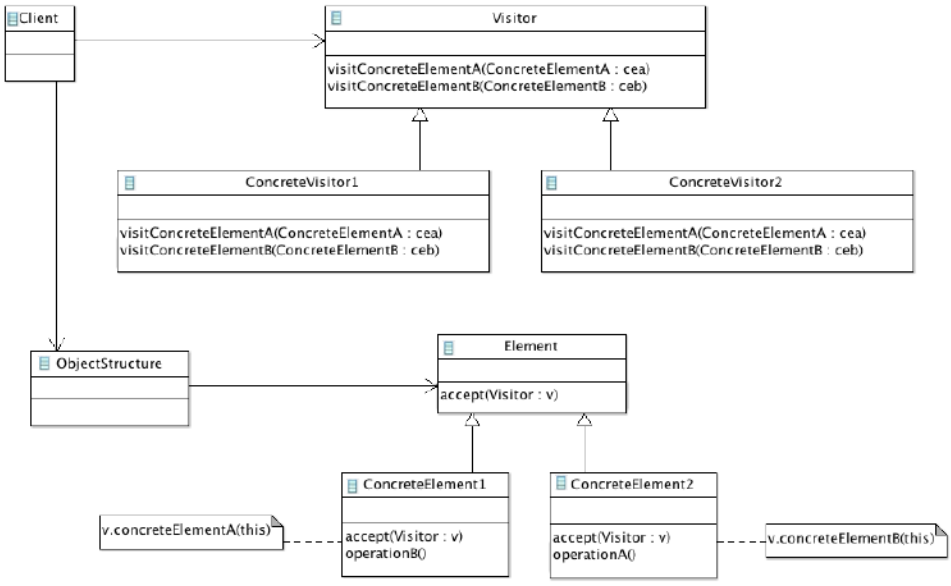


Fig. 35. Patrón de diseño *Visitor*

Strategy

Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.

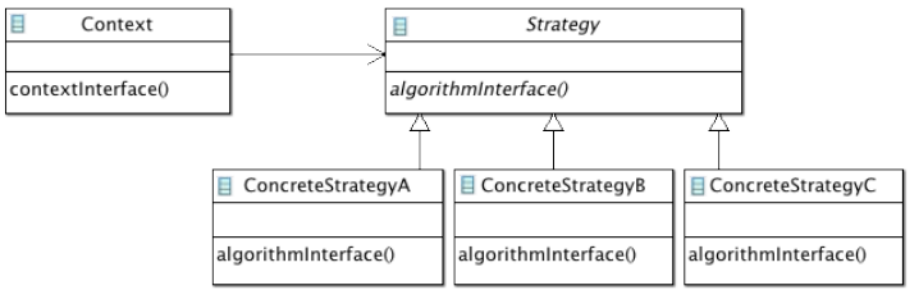


Fig. 36. Patrón de diseño *Strategy*

Template Method

Define, en una operación, el esqueleto de un algoritmo, y delega en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.

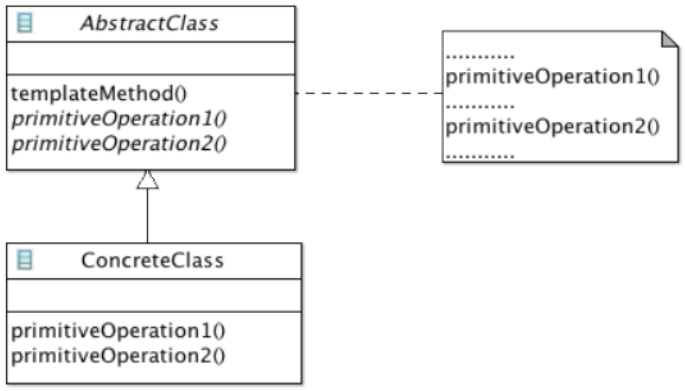


Fig. 37. Patrón de diseño *Template Method*

Template View

Una plantilla (*template*) es un documento tradicional de HTML con marcadores embebidos que son reemplazados, manipulados o evaluados a través de la API del motor de plantillas para producir un documento de salida.

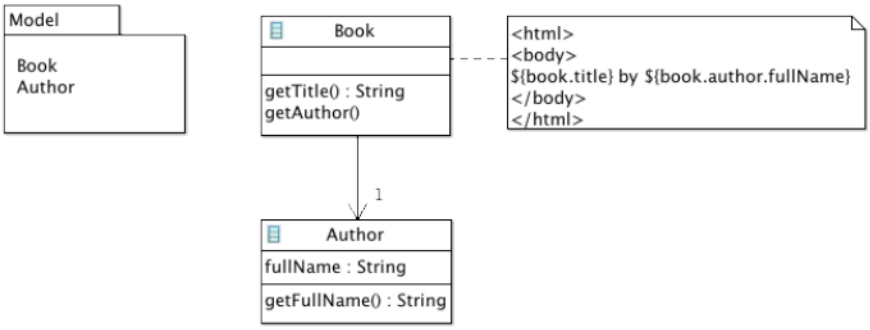


Fig. 38. Patrón de diseño *Template View*

En la próxima sección, se presenta cómo fueron aplicados en la implementación cada uno de los patrones que tuvieron lugar en esta sección.

5.2 IMPLEMENTACIÓN

La implementación se llevó a cabo en JAVA dado que la herramienta de soporte de *WebSpec* está escrita en JAVA. Se creó un módulo nuevo en el cual se abstrae todo lo concerniente a la generación de escenarios. Este módulo expone una API por medio de una interfaz JAVA (*UserStoryGenerator*), el único método que se expone se llama *generate,* y retorna un objeto (*UseStoryGenerationResponse*) con la respuesta del proceso de generación. En la Fig. 39, se muestra la interfaz y sus implementaciones.

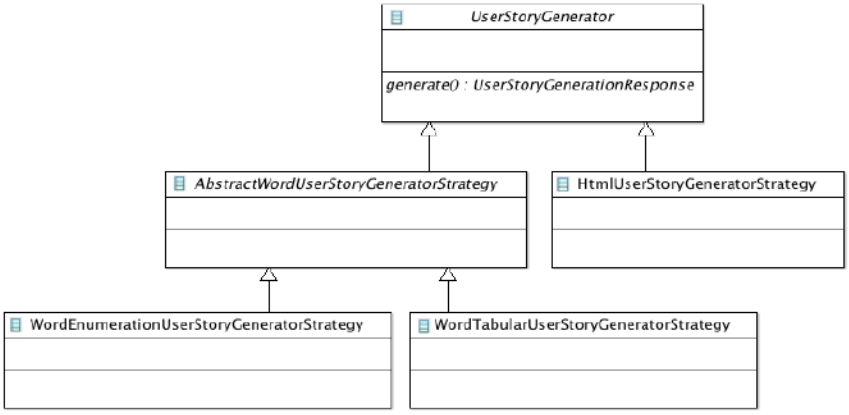


Fig. 39. Interfaz *UserStoryGenerator*

Como se puede apreciar en la Fig. 39, la interfaz *UserStoryGenerator* define un método *generate*. Éste recibe como parámetros un diagrama, un mapa, un archivo y una localización (por motivos de espacio, no están incluidos

en la imagen). El mapa contiene asociaciones entre interacciones, navegaciones y sus respectivas localizaciones en el diagrama; el archivo contiene una imagen del diagrama, y la localización nos dice qué lenguaje debemos utilizar para los textos. Como respuesta, el método *generate* retorna un objeto *UserStoryGenerationResponse*, el cual contiene todo lo necesario como para poder guardar el documento en memoria no volátil (Fig. 40).

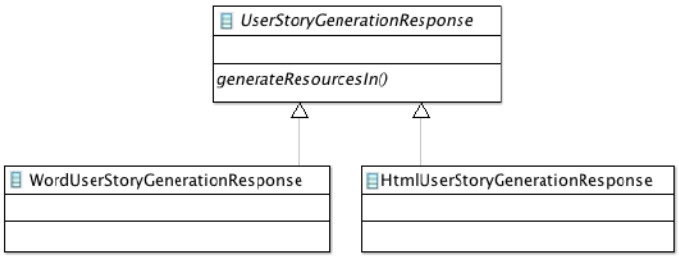


Fig. 40. Interfaz *UserStoryGenerationResponse*

Existen tres implementaciones concretas para la generación de escenarios: dos implementaciones basadas en WORD (*WordEnumerationUserStoryGeneratorStrategy)* y *WordTabularUserStoryGeneratorStrategy*), y una en HTML (*HtmlUserStoryGeneratorStrategy*).

La implementación HTML utiliza una librería por la cual podemos crear plantillas e ir completándolas (esta librería hace uso del patrón *Template View*).

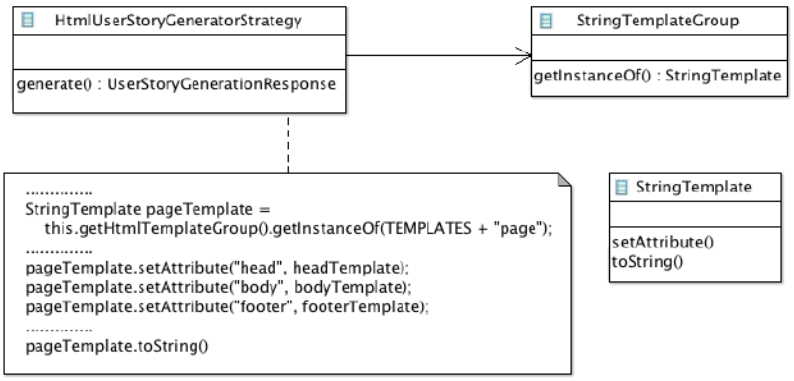


Fig. 41. *HtmlUserStoryGeneratorStrategy*

Las implementaciones WORD utilizan una librería para generar el documento WORD. Estas dos implementaciones hacen uso del patrón *Template Method*; hay una clase abstracta que define el algoritmo de generación, y en las subclases se definen ciertos pasos dependiendo de la estrategia (Fig. 42).

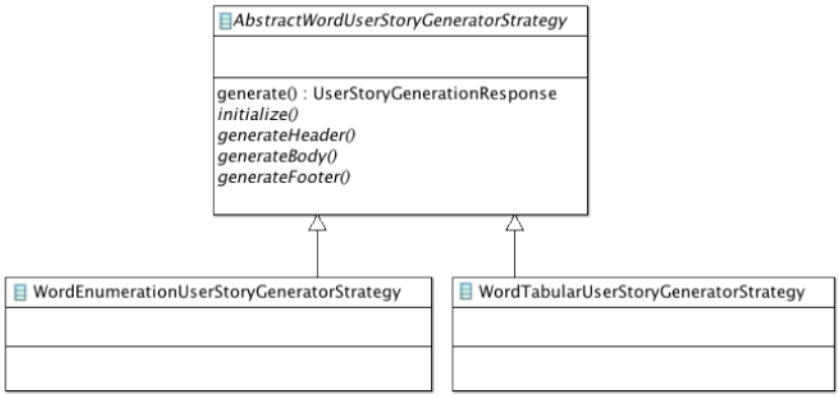


Fig. 42. Patrón *Template Method* aplicado

Si bien hay tres estrategias de generación de documentos, solo dos son distintas en cuanto a la estructura interna del documento: dos estrategias basadas en la generación de documentos WORD y una en HTML. La opción HTML contiene una estructura tabular, mientras que las opciones WORD

pueden contener una estructura tabular o enumerativa. Para poder obtener el contenido del documento (como se especifica en la Sección 4.4), es necesario recorrer el diagrama que recibimos como parámetro: éste tiene una estructura de grafo, por lo cual podemos recorrerlo con un algoritmo de recorrido (DFS). Para la generación del documento, necesitamos traducir cada nodo o arista del grafo de una manera determinada: cuando quiero su imagen del diagrama, cuando quiero su explicación textual o su *mockup* asociado, si es que corresponde. Para poder realizar esto, se implementó el patrón *Visitor*. En la Fig. 43, se puede ver la jerarquía creada para recorrer el diagrama y poder obtener las imágenes, las explicaciones y los *mockups* de las interacciones.

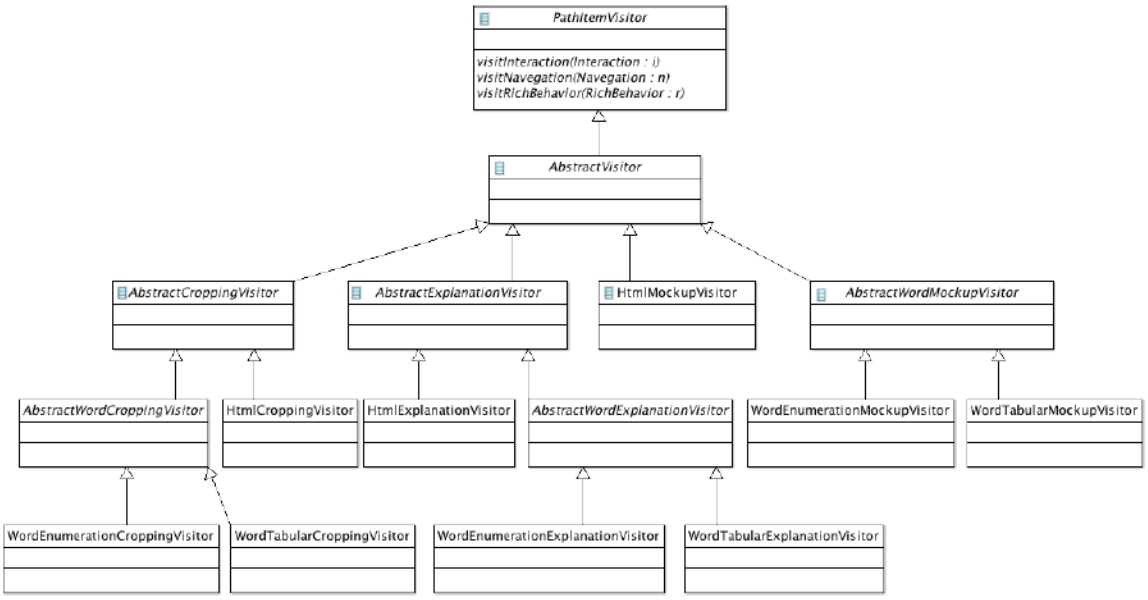


Fig. 43. Patrón *Visitor* aplicado

Una observación muy importante acerca de las clases concretas de la jerarquía presentada en la Fig. 43 es que son encargadas de ir generando las instrucciones específicas para poder obtener las imágenes y explicaciones. Estas instrucciones específicas dependen de la estrategia de generación que se haya elegido. No es lo mismo generar la explicación textual para la estrategia HTML en la cual se tiene que utilizar etiquetas HTML que generar la explicación

textual para la estrategia WORD, la cual tiene que ir agregando etiquetas de numeración y párrafos.

Para agregar la imagen de la interacción o navegación, se utilizan las siguientes clases de acuerdo con la estrategia elegida:

- *HtmlCroppingVisitor*
- *WordEnumerationCroppingVisitor*
- *WordTabularCroppingVisitor*

Para agregar la descripción textual de la interacción o navegación, se utilizan las siguientes clases según la estrategia elegida:

- *HtmlExplanationVisitor*
- *WordEnumerationExplanationVisitor*
- *WordTabularExplanationVisitor*

Para agregar la imagen del *mockup* asociado a la interacción, se utilizan las siguientes clases de acuerdo con la estrategia elegida:

- *HtmlMockupVisitor*
- *WordEnumerationMockupVisitor*
- *WordTabularMockupVisitor*

En la próxima sección, se presentan comentarios sobre las decisiones que se fueron tomando a lo largo de la etapa de implementación.

5.3 TECNOLOGÍAS

En esta sección, se hace una breve mención a aquellas tecnologías que fueron elegidas y utilizadas en la fase de implementación.

Docx4j

Es una librería para descomprimir un "paquete" .docx (*WordprocessingML*, *SpreadsheetMLPackage*, *PresentationMLPackage*) y analizar el código XML para crear una representación en memoria. Es de código abierto y disponible bajo la licencia de *Apache* (v2). Es similar al concepto de Microsoft OpenXML SDK para .NET. *Docx4j* depende en gran medida de *JAXB*, la arquitectura JAVA para manejar XML. Las partes relevantes de *docx4j* se generan a partir de los esquemas ECMA.

¿Qué cosas se pueden hacer con *docx4j*?

- Abrir archivos .docx (de sistemas de archivos: *SMB*/*CIFS*, *WebDAV* usando *VFS*), .pptx, .xlsx
- Crear nuevos archivos .docx, .pptx, .xlsx
- Manipular programáticamente los tipos de archivos anteriores

Se puede encontrar más información de esta librería en: http://www.docx4java.org/trac/docx4j

StringTemplate

Es un motor de plantillas (*templates*) para la generación de código fuente, páginas WEB, correos electrónicos o cualquier otro texto de salida formateado. ST es en sí un lenguaje de dominio específico para la generación de texto estructurado a partir de estructuras de datos internas. Algunas de sus características son:

- Herencia de plantillas de grupo

- Polimorfismo de plantillas
- Evaluación retardada
- Recursividad
- La producción de auto sangría
- Interfaces de grupo
- Regiones de plantillas

La experiencia muestra que es fácil de aprender y que satisface las necesidades. La principal contribución de ST es la clara identificación e implementación de una solución consistente y robusta al problema de renderizado de estructuras de datos a texto, sin descuidar la naturaleza de la generación en sí y el objetivo fundamental de mantener la separación entre la lógica de generación y las plantillas de salida.

Se puede encontrar más información de esta librería en: http://www.stringtemplate.org/

CAPÍTULO 6

CASO DE ESTUDIO

El objetivo de este capítulo es utilizar *WebSpec* en un contexto real de la industria de software, además de poder obtener conclusiones que permitan enriquecer *WebSpec*. Es importante destacar que el principal foco de atención de este caso de estudio es la derivación de los diagramas de *WebSpec*.

La organización del capítulo consta de una primera sección en la cual se hace una breve introducción (Sección 6.1); en la siguiente sección, se presenta el contexto real en el cual se utilizó *WebSpec* (Sección 6.2); posteriormente, se presenta un seguimiento detallado de la herramienta de derivación de *WebSpec* sobre el contexto real de aplicación (Sección 6.3); y, finalmente, se presentan algunas observaciones (Sección 6.4).

6.1 INTRODUCCIÓN

Se ha utilizado *WebSpec* para asistir en el desarrollo de un nuevo conjunto de requerimientos de una aplicación que ya se encontraba en un ambiente productivo. En este conjunto de requerimientos, el requerimiento clave para poder llevar adelante los restantes fue la posibilidad de integrarse con distintas redes sociales. El hecho de que la aplicación en cuestión no provee la posibilidad de que el usuario pueda registrarse y completar su perfil, sumado a la reticencia de los usuarios a registrarse en una nueva página, hizo que se

pensara en la posibilidad de obtener los datos necesarios del usuario de alguna red social. De esta manera, el usuario no tiene que completar extensos formularios de registración, ni dejar datos sensibles en una nueva página, ni perder tiempo realizando lo anteriormente mencionado. En la próxima sección, se presentará con mayor detalle la aplicación en cuestión y la nueva funcionalidad.

6.2 CONTEXTO

La aplicación en la cual se utilizó *WebSpec* para llevar adelante el desarrollo de este nuevo conjunto de requerimientos se llama *MiDespegar*. Ésta es la encargada de facilitar una serie de acciones que el usuario puede realizar luego de realizar la compra de un producto. Una de las limitaciones que tenía *MiDespegar* antes de este desarrollo era que el usuario no podía ver todos sus productos comprados, por lo cual tenía que ingresar con el número de compra del producto. Luego del nuevo desarrollo, el usuario puede ingresar por medio de una red social, y *MiDespegar* le muestra todos sus productos comprados.

A continuación, enumeramos una serie de historias de usuario que *MiDespegar* debía cumplir luego de la implementación de los requerimientos:

- El usuario puede ingresar por medio de una red social y ver todos sus productos comprados.
- El usuario puede ver el detalle de cada uno de sus productos y realizar acciones sobre los mismos.
- El usuario puede agregar un nuevo correo electrónico con el cual haya comprado un producto.
- El usuario puede validar el agregado de un nuevo correo electrónico por medio de un link con un *token* asociado que le llega al correo electrónico.

- El usuario puede acceder a *MiDespegar* desde el correo electrónico que se le envía luego de haber realizado una compra.
- El usuario puede acceder a *MiDespegar* desde la página de gracias que obtiene luego de haber realizado una compra.

De las historias de usuario enumeradas anteriormente se derivó en una serie de tareas por realizar. Para llevar adelante todo el desarrollo, se agruparon las tareas obtenidas en iteraciones de una semana. Se determinaron cuatro iteraciones, con lo cual, al cabo de cuatro semanas, todas las nuevas características funcionales estaban completas. La agrupación de tareas quedó como se muestra a continuación.

Iteración 1

- Desarrollo del componente social para la interacción con redes sociales
- Desarrollo del módulo de interacción con Facebook
- Desarrollo del módulo de interacción con Google

Iteración 2

- Desarrollo de la UI y *backend* para mostrar el listado de los productos comprados
- Integración con la fuente de datos de productos comprados (CRO)
- Integración con la fuente de datos de productos comprados (FV)
- Integración con la fuente de datos de productos comprados (FH)

Iteración 3

- Desarrollo del componente que almacena los correos electrónicos que el usuario va agregando
- Desarrollo del flujo de validación del agregado de correos electrónicos

Iteración 4

- Desarrollo de servicios para que aplicaciones externas puedan redirigir la navegación hacia *MiDespegar*.

6.3 *WEBSPEC* EN USO

En esta sección, se muestra cómo se utilizó *WebSpec* con la nueva herramienta de generación de escenarios. Para ello, se seleccionaron dos escenarios que derivan del escenario “El usuario puede acceder a *MiDespegar* desde la página de gracias que obtiene luego de haber realizado una compra” enumerado en la Sección 6.2

- Luego de haber realizado una compra, el usuario quiere ver sus reservas compradas.
- Luego de haber realizado una compra, el usuario quiere agregar un correo electrónico con el cual realizó una compra previamente.

Se comenzó por construir el diagrama de *WebSpec* que representa estos dos escenarios. La construcción se llevó a cabo por medio de un proceso iterativo en el cual participaron un desarrollador, un analista funcional y un analista de usabilidad. En la Fig. 44, se muestra el diagrama obtenido luego de varias iteraciones.

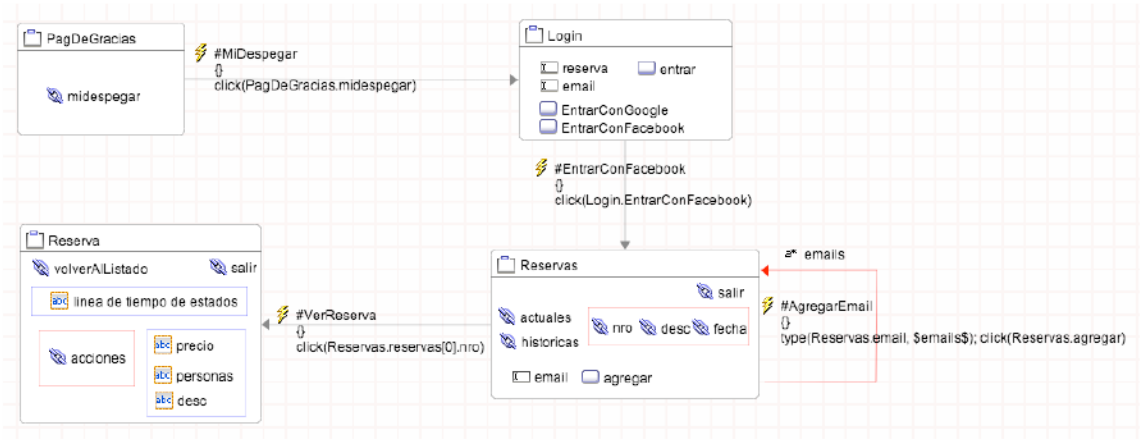


Fig. 44. Diagrama de *WebSpec* obtenido

A medida que se fue construyendo el diagrama de la Fig. 44, se fueron construyendo los *mockups* de las interacciones. Por medio de lo *mockups,* se pudo realizar una simulación de cómo se comportaría la aplicación una vez implementados los escenarios. La simulación fue útil para validar la construcción del diagrama. En la Fig. 45, se ven los *mockups* correspondientes a las interacciones del diagrama de la Fig. 44.

Fig. 45. *Mockups* de las interacciones

Por otro lado, se generó una serie de documentos para documentar y comunicar los diagramas que se fueron construyendo a lo largo de las iteraciones. Estos documentos se generaron por medio de la herramienta que se desarrolló para esta tesis. A continuación, se muestra parte de los documentos generados. En la Fig. 46, se muestra el documento HTML con el escenario "Luego de haber realizado un compra, el usuario quiere agregar un correo

electrónico con el cual realizó una compra previamente"; luego, en la Fig. 47, se muestra el documento WORD con la estrategia enumerativa y el escenario "Luego de haber realizado una compra, el usuario quiere ver sus reservas compradas"; y finalmente, en la Fig. 48, se muestra el documento WORD con la estrategia tabular y el escenario "Luego de haber realizado un compra, el usuario quiere agregar un correo electrónico con el cual realizó una compra previamente".

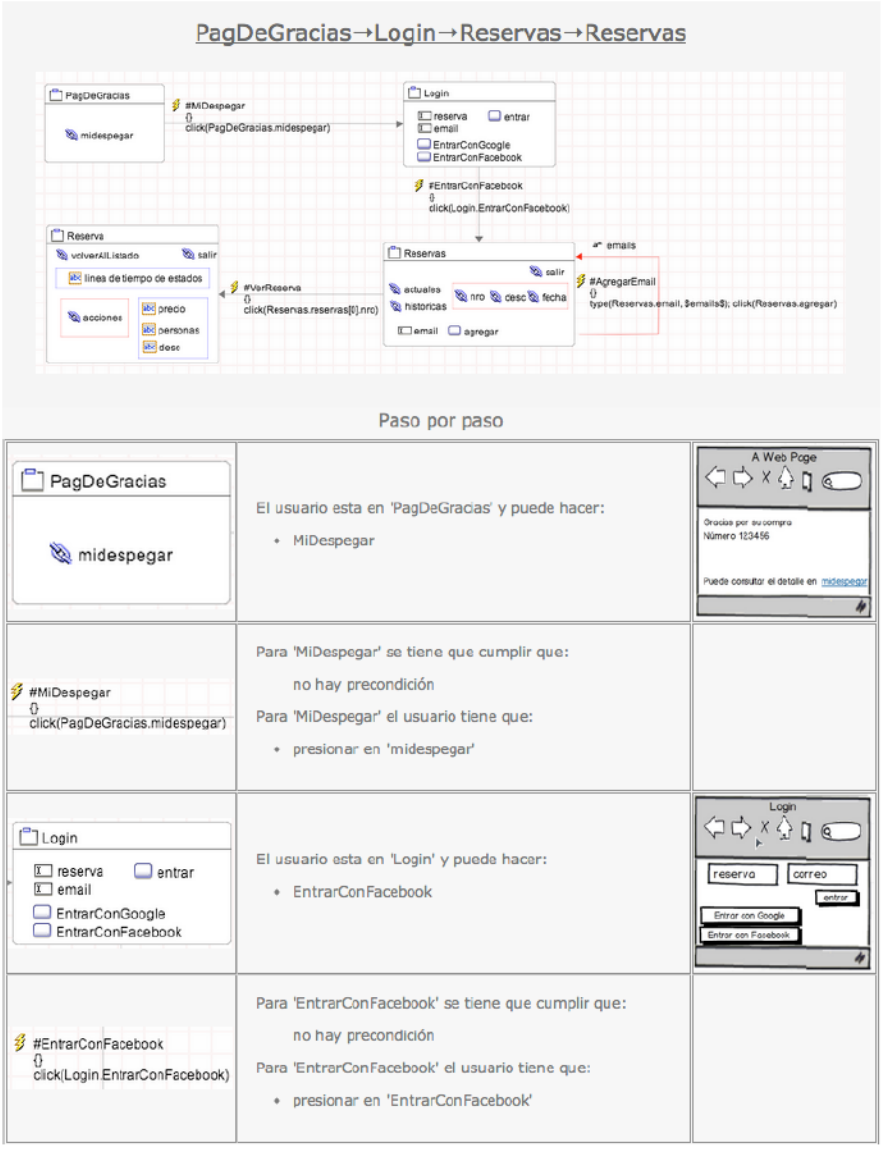
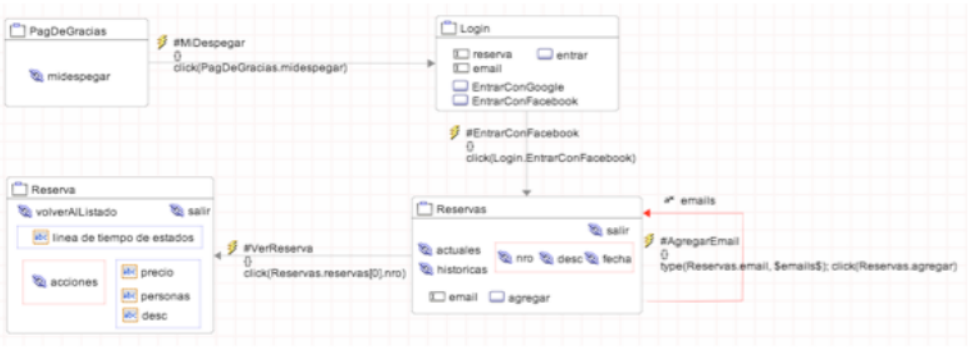


Fig. 46. Parte del documento HTML

2. PagDeGracias--> Login--> Reservas--> Reserva



2.1 Interacción 'PagDeGracias'

2.1.1 Diagrama de interacción 'PagDeGracias':

2.1.2 Explicación del diagrama de interacción 'PagDeGracias':

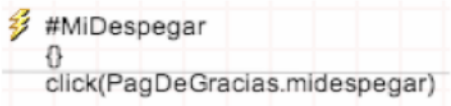
El usuario esta en 'PagDeGracias' y puede hacer:

- 'MiDespegar'

2.1.3 Mockup del diagrama de interacción 'PagDeGracias':

2.2 Navegación 'MiDespegar'

2.2.1 Diagrama de navegación 'MiDespegar':



2.2.2 Explicación del diagrama de navegación 'MiDespegar':

Para 'MiDespegar' se tiene que cumplir que:

- no hay precondición

Para 'MiDespegar' el usuario tiene que:

- presionar en 'midespegar'

2.3 Interacción 'Login'

2.3.1 Diagrama de interacción 'Login':

2.3.2 Explicación del diagrama de interacción 'Login':

El usuario esta en 'Login' y puede hacer:

- 'EntrarConFacebook'

2.3.3 Mockup del diagrama de interacción 'Login':

2.4 Navegación 'EntrarConFacebook'

2.4.1 Diagrama de navegación 'EntrarConFacebook':

2.4.2 Explicación del diagrama de navegación 'EntrarConFacebook':

Para 'EntrarConFacebook' se tiene que cumplir que:

- no hay precondición

Para 'EntrarConFacebook' el usuario tiene que:

- presionar en 'EntrarConFacebook'

2.5 Interacción 'Reservas'

2.5.1 Diagrama de interacción 'Reservas':

2.5.2 Explicación del diagrama de interacción 'Reservas':

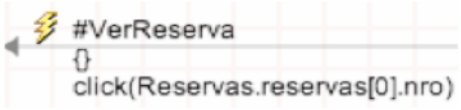
El usuario esta en 'Reservas' y puede hacer:

- 'AgregarEmail'
- 'VerReserva'

2.5.3 Mockup del diagrama de interacción 'Reservas':

2.6 Navegación 'VerReserva'

2.6.1 Diagrama de navegación 'VerReserva':



2.6.2 Explicación del diagrama de navegación 'VerReserva':

Para 'VerReserva' se tiene que cumplir que:

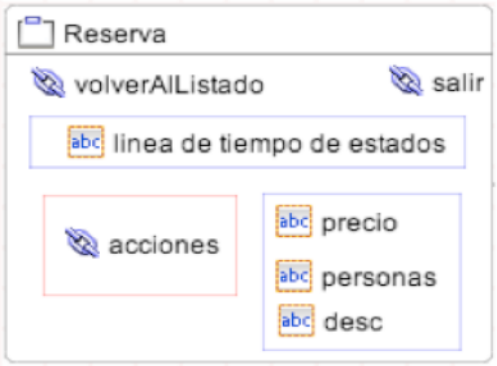
- no hay precondición

Para 'VerReserva' el usuario tiene que:

- presionar en 'nro'

2.7 Interacción 'Reserva'

2.7.1 Diagrama de interacción 'Reserva':



2.7.2 Explicación del diagrama de interacción 'Reserva':

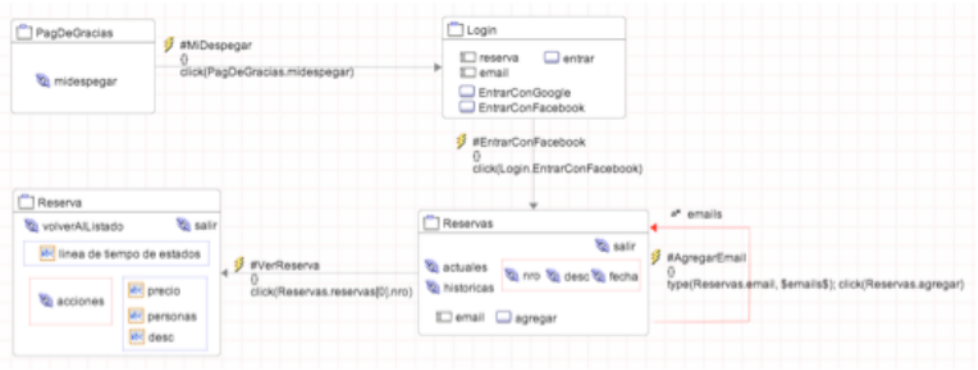
El usuario esta en 'Reserva' y puede hacer:

- no hay acciones

2.7.3 Mockup del diagrama de interacción 'Reserva':

Fig. 47. Parte del documento WORD: estructura enumerativa

1. PagDeGracias--> Login--> Reservas--> Reservas



Step	Explanation	Mockup
	The user is in 'PagDeGracias' and is able to do: • 'MiDespegar'	
	To 'MiDespegar' must be satisfied that: • there is no precondition To 'MiDespegar' the user has to perform next actions: • click on 'midespegar'	

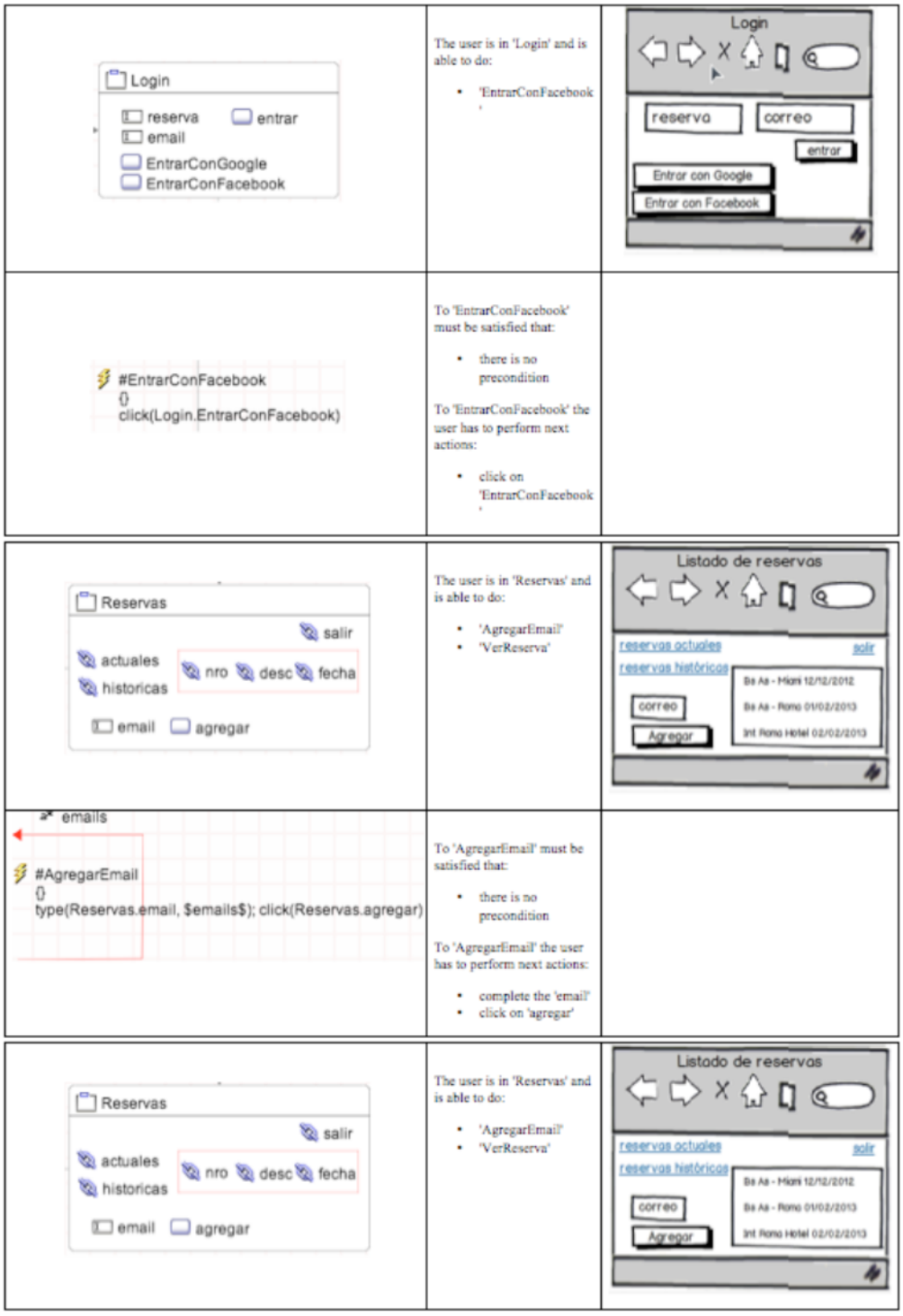


Fig. 48. Parte del documento WORD: estructura tabular

Finalmente, mostramos imágenes de la aplicación luego de finalizadas las cuatro iteraciones. Cabe mencionar que este conjunto de requerimientos implementados se encuentra en el ambiente productivo.

En la Fig. 49, se puede ver claramente cuál es la página que obtiene el usuario luego de haber realizado una compra. En la parte inferior derecha, se encuentra un párrafo en donde se informa que el usuario puede seguir su reserva y realizar modificaciones a través de *MiDespegar*.

En la Fig. 50, se puede ver la página de entrada a *MiDespegar*; en este punto, tenemos la posibilidad de entrar por medio de una red social o con los datos de una reserva en particular.

En la Fig. 51, se puede ver el listado de reservas asociadas al usuario de la red social; para esto, se toma el correo electrónico como referencia, además el usuario puede agregar un nuevo correo electrónico con el que haya realizado una compra previamente.

En la Fig. 52, se muestra el detalle de la reserva; en este punto, podemos observar los estados por los cuales pasó la reserva, el detalle de ésta y una lista de acciones que el usuario puede llevar a cabo.

Fig. 49. Página de gracias

Fig. 50. Página de entrada a *MiDespegar*

Fig. 51. Página con el listado de reservas

Fig. 52. Página donde se muestra el detalle de la reserva

6.4 OBSERVACIONES

En esta última sección, se presentan conclusiones y comentarios relevantes sobre el caso de estudio presentado a lo largo del presente capítulo.

Luego de culminar las cuatro iteraciones que se propusieron para llevar adelante el desarrollo de los nuevos requerimientos, se recolectó información acerca de la utilización de *WebSpec*.

Desde el punto de vista de los integrantes del equipo de producto (analista funcional y analista de usabilidad), la aceptación del uso de la herramienta superó las expectativas. Generalmente, se está acostumbrado a trabajar con herramientas con las cuales se tiene alguna experiencia o conocimiento previo para realizar el trabajo diario, en vez de incursionar en la utilización de una nueva herramienta o tecnología. Con una mínima introducción sobre los diagramas, los analistas empezaron a probar *WebSpec* y asimilar su funcionamiento. Las herramientas de simulación y generación de escenarios fueron un importante soporte para validar la construcción de los diagramas que

iban realizando. Por otro lado, se imprimieron algunos documentos de los diagramas para poder discutir sobre el mismo papel, así como para validar la viabilidad de los escenarios con los desarrolladores.

Desde el punto de vista de los desarrolladores, la aceptación fue la esperada. Esto se debe a que los desarrolladores estaban mucho más relacionados con los conceptos de navegación en aplicaciones WEB y construcción de páginas con comportamiento rico.

Si bien la aceptación de la herramienta fue positiva, tenemos que tener en consideración que todas las personas involucradas en el proceso de desarrollo son informáticos. En este caso de estudio particular, los analistas en su momento fueron desarrolladores de software que, con el tiempo, migraron a puestos funcionales. Lo ideal hubiese sido que el mismo usuario ayudara o construyera los diagramas; claro está que este escenario es casi imposible dada la naturaleza de la empresa para la cual se realizó el desarrollo.

CAPÍTULO 7

CONCLUSIONES Y TRABAJO FUTURO

En este capítulo final, se resume el trabajo presentado en esta tesis. Principalmente, se hace mención a las contribuciones realizadas (Sección 7.1) y a las limitaciones del presente trabajo (Sección 7.2). Finalmente, la Sección 7.3 presenta posibles extensiones y trabajos futuros.

7.1 CONTRIBUCIÓN

Desde un punto de vista global y general, el aporte que se realiza por medio de esta tesis contribuye indirectamente al desarrollo de aplicaciones WEB. Como se ha mencionado en los capítulos iniciales, la ingeniería de requerimientos tiene un rol determinante en el proceso de desarrollo de aplicaciones WEB. *WebSpec* fue presentado como una herramienta para llevar adelante la etapa de ingeniería de requerimientos (obtención, especificación y validación de requerimientos) del proceso de desarrollo de aplicaciones WEB.

La contribución principal viene acompañada por *WebSpec*: ésta permite especificar características fundamentales de los requerimientos de las aplicaciones WEB (navegación e interacción).

Por medio de la implementación realizada en esta tesis, *WebSpec* posee una característica más, cuyo objetivo es que el usuario cuente con una alternativa para poder ir comprendiendo los diagramas que va realizando y

pueda también obtener un documento que pueda ser utilizado como documentación y medio de comunicación. Los documentos con formato tabular (HTML y WORD) son apropiados como comprensión y comunicación de los diagramas, y el formato enumerativo presenta una mejor estructura desde el punto de vista de la documentación.

Por lo tanto, *WebSpec* se transforma en una herramienta mucho más consistente y efectiva a la hora de su utilización en el proceso de ingeniería de requerimientos de aplicaciones WEB.

7.2 LIMITACIONES

Considerando que no es posible cubrir todos los aspectos de la transformación de un modelo a texto, existen algunas limitaciones en esta primera versión.

Limitaciones de la alternativa con WORD y estructura tabular:

- **Manejo de imágenes**: Si las imágenes insertadas son muy grandes, éstas toman un tamaño fijo para no romper el formato del documento.
- **Manejo de tablas**: Dado el tamaño de las imágenes, solamente se pueden insertar dos filas por hoja en el documento.
- **Formato de las páginas**: Dado que la imagen del escenario tiende a ser grande, la orientación de las páginas tiene que ser horizontal.

Limitaciones de la alternativa con WORD y estructura enumerativa:

- **Manejo de imágenes**: Si las imágenes insertadas son muy grandes, éstas toman un tamaño fijo para no romper el formato del documento.

La principal limitación de los documentos WORD es la utilización de las imágenes. Una vez generados los documentos, las imágenes pueden ser reajustadas en su tamaño como para que el documento quede acorde y pueda ser transferido o impreso. Esta principal limitación fue la motivación para ofrecer otro tipo de documento (HTML) en el cual se pueda abordar esta limitación.

7.3 TRABAJO FUTURO

A continuación, se describe el trabajo que, por cuestiones de tiempo y de alcance de esta tesis, no fue abarcado:

- Mejorar el manejo de imágenes de los documentos de tipo WORD.
- Mejorar la estructura tabular de los documentos de tipo WORD.
- Ofrecer un nuevo tipo de documento, por ejemplo: diapositivas.
- Ofrecer la posibilidad de generar documentos de tipo PDF.
- Obtener los documentos en nuevos idiomas, por ejemplo: alemán y portugués.

REFERENCIAS

[AKS03] Agrawal, A.; Karsai, G. y F. Shi (2003), Graph Transformations on Domain-Specific Models, Technical Report ISIS-03-403, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37203, 2003.

[ASP] Aspose, File formats components, último acceso 2012. http://www.aspose.com/

[AXW12] Axure, a wireframing, rapid prototyping, and specification software tool aimed at web and desktop applications, último acceso 2012. http://www.axure.com

[BAL12] Balsamiq, a graphical user interface mockup builder application, último acceso 2012. http://www.balsamiq.com/products/*mockups*

[BRJ99] Booch, G.; Rumbaugh, J. y I. Jacobson (1999), Unified Modeling Language User Guide, AddisonWesley.

[BTDD2] Beck, K. (2002), Test Driven Development: By Example (Addison-Wesley Signature Series).

[BXP00] Beck, K. (2010), "Extreme Programming Explained": Embrace Change, Addison-Wesley, Pearson Education, 2000, ISBN 201-61641-6.

[CFBB03] Ceri, S.; Fraternali, P.; Bongio, A.; Brambilla, M.; Comai, S. y M. Matera (2003), Designing Data-Intensive Web Applications, Morgan Kaufman.

[CH00] Claessen, K. y J Hughes (2000), QuickCheck: a lightweight tool for random testing of Haskell programs. In: Proceedings of the fifth ACM SIGPLAN international conference on functional programming, vol 35, pp 268–279.

[DOC4J] Docx4j, a Java library for creating and manipulating Microsoft Open XML (Word docx, Powerpoint pptx, and Excel xlsx) files, último acceso 2012. http://www.docx4java.org

[DOCMS] Docmosis, a simple and fast solution for generating MS Word, PDF or Open Office documents from your business templates, http://www.docmosis.com/, último acceso 2012.

[DTL97] De Troyer, O. y C. Leune (1997), WSDM: A User Centered Design Method for Web Sites, Technical Report of Tilburg University, Infolab, Belgium.

[DVAP04] Patariczа, A. y D. Varró (2004), Generic and Meta-Transformations for Model Transformation Engineering, Proceedings of the 7th International Conference on the Unified Modeling Language, Lisbon, Portugal, pp. 290–304, 2004.

[ECMA] ECMA, Standard defines the ECMAScript scripting language, último acceso 2012. http://www.ecma-international.org/publications/standards/Ecma-262.htm

[EDW03] Willink, E. D. (2003), UMLX: A Graphical Transformation Language for MDA, Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Anaheim, CA(2003), pp. 13–24, 2003.

[EEKR99] Ehrig, H.; Engels, G.; XKreowski, H. J. y G. Rozenberg (eds.), (1999), Handbook on Graph Grammars and Computing by Graph Transformation, vol. 2: Applications, Languages and Tools, World Scientific, 1999.

[EMF12] Eclipse EMF, a modeling framework and code generation facility for building tools and other applications based on a structured data model, último acceso 2012. http://www.eclipse.org/emf/

[ET02] Escalona, M.J.; Mejías, M. y J. Torres (2002), Methodologies to develop Web Information Systems and Comparative Analysis, Informatik/Informatique, núm. 2/2002 de I/I.

[ETM02] Escalona, M.J.; Mejías, M. y J. Torres (2002), Requirements Capture Workflow in Global Information Systems, Proceedings of OOIS, Springer-Verlag, Montpellier, France.

[FTL12] Freemarker, a generic tool to generate text output (anything from HTML to autogenerated source code) based on templates, último acceso 2012, http://freemarker.sourceforge.net/

[FTS12] Fujaba Tool Suite 4, University of Paderborn Software Engineering, último acceso 2012, http://www.fujaba.de

[GCP00] Cachero, C.; Gómez, J. y O. Pastor (2000), Extending a conceptual modeling approach to web application design, En actas de: Conference on Advanced Information Systems Engineering (CAISE' 00), Springer Verlag, LNCS 1789, pp. 79–93.

[GHJV95] Gamma, E.; Helm, R.; Johnson, R. y J. Vlissides (1995), Design patterns: elements of reusable object-oriented software, Addison-Wesley Longman Publishing Co, Boston.

[GMF12] Eclipse GMF, a set of generative components and runtime infrastructures for developing graphical editors based on EMF and GEF, último acceso 2012, http://www.eclipse.org/modeling/gmp/

[J2W12] Java2word, Java library for generation of Word documents, último acceso 2012, http://code.google.com/p/java2word/

[JAXB12] JAXB, the Java Architecture for XML Binding, último acceso 2012, http://jaxb.java.net/

[JK05] Jouault, F. y I. Kurtev (2005), Transforming Models with ATL, In Proc. of the Model Transformations in Practice Workshop at MoDELS 2005, Jamaica, 2005.

[JLHV02] de Lara, J. y H. Vangheluwe (2002), AToM: A Tool for Multi-Formalism and Meta-Modeling, Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering, France, pp. 174–188, 2002.

[JQY12] jQuery, the write less, do more, JavaScript library, último acceso 2012, http://jquery.com/

[JSP12] JSP, technology provides a simplified, fast way to create dynamic web content, último acceso 2012. http://www.oracle.com/technetwork/java/javaee/jsp/index.html

[K99] Koch, N. (1999), A Comparative Study of Methods for Hypermedia Development, Technical Report 9905, Ludwig-Maximilian-University, Munich, Germany.

[KBC04] Barzdins, J.; Celms, E. y A. Kalnins (2004), Model Transformation Language MOLA, Proceedings of Model Driven Architecture: Foundations and Applications, Linkoeping, Sweden, pp. 14-28,

2004.

[MDWE] Giandini, R.; Pérez, G. y C. Pons (2010), Desarrollo de Software Dirigido por Modelos.

[MFPEA] Fowler, M. (2002), Patterns of Enterprise Application Architecture, Addison-Wesley.

[MOD09] Moody, D. (2009) The physics of notations: toward a scientific basis for constructing visual notations in software engineering, IEEE Trans Softw Eng 35(6):756–779. doi:10.1109/TSE.2009.67.

[MW03] Maximilien, E. M. y L. Williams (2003), Assessing test-driven development at IBM, In: Proceedings of the 25th international conference on software engineering (Portland, Oregon, May 03–10, 2003), International conference on software engineering, IEEE Computer Society, Washington, DC, pp 564–569.

[PAF01] Abrahão, S.; Fons, J. y O. Pastor (2001), An Object-Oriented Approach to Automate Web Applications Development. In: Proceedings of the Internacional Conference of Electronic Commerce and Web Technologies (EC-Web 2001), Springer Verlag, LNCS2115, pp. 16-28.

[PMM97] Mancini, C.; Meniconi, S. y F. Paternó; (1997), ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, Actas de Human-Computer Interaction (INTERACT '97), Chapman & Hall, pp. 362-366.

[POI12] POI, API for manipulating various file formats based upon the Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2),
último acceso 2012. http://poi.apache.org/hwpf/index.html

[QVT12] QVT, Query/View/Transformation, último acceso 2012,
http://www.omg.org/spec/QVT/

[RBGRL] Burella, J.; Grigera, J.; Robles Luna, E. y G. Rossi (2010), Incremental Usability Improvement in an Agile Approach for Web Applications.

[RLRG1] Garrigós, I.; Robles Luna, E. y G. Rossi (2011),
WebSpec a visual language for specifying interaction and navigation requirements in web applications.

[RRLIG] Garrigós, I.; Robles Luna, E. y G. Rossi (2010),

Capturing and Validating Personalization Requirements in Web Applications.

[SBM01] Beedle, M.; Martin, R. C. y K. Schwaber (2001), “Agile Software Development with SCRUM”, Prentice Hall.

[SEL12] Selenium, Web application testing system, último acceso 2012, http://www.seleniumhq.org/.

[SR98] Rossi, G. y D. Schwabe (1998), Developing Hypermedia Applications using OOHDM. Workshop on Hypermedia Development Process, Methods and Models, Hypertext´98, Pittsburg, USA.

[STT12] StringTemplate, a java template engine (with ports for C#, Python) for generating source code, web pages, emails, or any other formatted text output, último acceso 2012, http://www.stringtemplate.org/

[TAG03] Taentzer, G. (2003), AGG: A Graph Transformation Environment for Modeling and Validation of Software, Application of Graph Transformations with Industrial Relevance (AGTIVE’03) 3062, pp. 446–453, 2003.

[UCIJ02] Bittner, K. y I. Spence (2002), Use Case Modelling (Addison-Wesley Object Technology).

[VEL12] Velocity, a Java-based template engine that provides a template language to reference objects defined in Java code, último acceso 2012, http://velocity.apache.org/

[VFP05] Fons, J.; Pelechano, V. y P. Valderas (2005), From Web Requirements to Navigational Design - A Transformational Approach, In: Proceedings of the Internacional Conference of Web Engineering (ICWE 2005), Springer Verlag, LNCS 3579, pp. 506-511.

[VSS00] Vilain, P.; Schwabe, D.; Sieckenius, C. (2000). A diagrammatic Tool for Representing User Interaction in UML. Lecture Notes in Computer Science. Proc. UML’2000. York, England.

[VVVP02] Pataricza, D.; Varró, A. y G. Varró (2002), Designing the Automatic Transformation of Visual Languages, Science of Computer Programming 44, No. 2, 205–227, 2002.

[WDR12] WebDriver, a tool for automating testing web applications, and in

particular to verify that they work as expected, último acceso 2012, http://www.webdriver.googlecode.com

[WRLG] Garrigós, I.; Grigera, J.; Robles Luna, E. y Winckler, M. (2010), Capture and Evolution of Web requirements using WebSpec.

[WTDD] Grigera, J.; Robles Luna, E. y Rossi, G. (2009), Bridging test and model-driven approaches in web engineering, In: Díaz, O.; Gaedke, M. y M. Grossniklaus (eds), Web Engineering, volume 5648 of Lecture Notes in Computer Science, pages 136–150, Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-02818-210.