Mockup-Driven Development: Introduciendo Agilidad en Procesos Basados en Modelos

Autor: José Matías Rivero

Directora: Dra. Silvia Gordillo

Co-Directores: Dr. Gustavo Rossi, Dr. Francisco Montero

Tesis presentada para obtener el grado de Doctor en Ciencias Informáticas

Facultad de Informática - Universidad Nacional de La Plata

- Octubre de 2014 -

Índice

1. Introducción

En los últimos años las metodologías de Model-Driven Web Engineering (MDWE) como WebML [1], UWE [2] o OOHDM [3] se han convertido en soluciones maduras para el desarrollo de Aplicaciones Web. Estas metodologías aplican prácticas de Desarrollo Dirigido por Modelos (Model-Driven Development o MDD) para capturar conceptos de software en modelos los cuales son utilizados a posteriori para obtener aplicaciones corrientes. En lugar de estar basados en codificación directa, los procesos de desarrollo MDD hacen énfasis en el uso de modelos de alto nivel para generar el código de la aplicación final. Los conceptos utilizables en los modelos que componen a cada proceso MDD particular están descriptos usando un *metalenguaje*, usualmente llamado *metamodelo*. Las metodologías MDWE representan procesos MDD orientados al desarrollo de Aplicaciones Web; por lo tanto, sus metamodelos están conformados por conceptos Web tales como páginas, links, comportamientos RIA [4], entre muchos otros. El proceso de desarrollo MDWE clásico consta de 3 pasos [5]: (1) la construcción de modelos de dominio o datos, (2) la definición de un modelo de hipertexto y (3) el refinado del mismo para definir el *look and feel* de la aplicación. El resultado del proceso es un conjunto de modelos a partir de los cuales pueden generar la Aplicación Web final a través de herramientas de generación de código.

Si bien los procesos MDWE clásicos mejoran la productividad en el desarrollo utilizando lenguajes de alto nivel para describir Aplicaciones Web, los mismos tienden a dejar aspectos importantes para los usuarios – como detalles de presentación e interacción de la Interfaz de Usuario (también denominada *User Interface* o UI) – para las etapas finales del ciclo de desarrollo [6]. En consecuencia, los clientes sólo ven su Aplicación Web funcionando después de una iteración completa, pudiendo recién en ese entonces descubrir nuevos requerimientos que supongan una reestructuración de partes de la misma. Los cambios en la UI no sólo pueden implicar aspectos de presentación, interacción y usabilidad, sino también la lógica de negocios de la aplicación [7,8]. En el contexto de MDWE, esto implica cambios potenciales en los modelos en los tres niveles.

Mientras que el uso de lenguajes de alto nivel facilita la transformación requerimientos en software funcional en las metodologías MDWE, las dependencias *duras* entre cada etapa de modelado ralentizan el proceso. Por ejemplo, si los usuarios finales requieren una nueva forma de introducir datos para un nuevo tipo de objeto, los desarrolladores deben primero construir los modelos de dominio, hipertexto y presentación hasta que el usuario vea y pueda probar efectivamente la interfaz de usuario a través de la cual podrá crear el mencionado objeto. En este contexto, los requerimientos descubiertos a través de (por ejemplo) reuniones directas con los usuarios finales deben persistirse en algún artefacto intermedio de requerimientos hasta que puedan ser traducidos a artefactos de especificación de software concretos. En adición a la necesidad de persistir requerimientos, los procesos MDWE actuales no proveen mecanismos claros para facilitar su trazabilidad ni encontrar conflictos en los mismos. Al igual que en las metodologías basadas en código, los requerimientos deben ser traducidos manualmente desde los artefactos de requerimientos iniciales – por ejemplo, Casos de Uso, User Stories, prototipos de UI, etc. Respecto a los conflictos o inconsistencias en requerimientos, las propuestas actuales sólo permiten descubrir errores estructurales en los modelos – y no en los conceptos relacionados al dominio en sí [9].

Para resolver esta problemática, el autor ha investigado varias de las prácticas emergentes en el contexto de las Metodologías Ágiles (dado que son actualmente las más utilizadas en la industria [10], especialmente en el campo de la Ingeniería de Requerimientos [11]). En este contexto, los prototipos de interfaz de usuario (generalmente conocidos como *mockups* o *wireframes*), han demostrado aumentar la eficiencia [8] en la captura de requerimientos de las Aplicaciones Web, volviéndose extensivamente usados [12–14]. Una de las ventajas inmediatas que proveen es el ser técnicamente valiosos para los desarrolladores y, al mismo tiempo, totalmente comprensibles por los usuarios finales [15]. El enfoque presentado en esta tesis se basa en el uso de mockups detallados (construidos a partir de User Stories [16]) para capturar la mayoría de los requerimientos relacionados con Aplicaciones Web y su arquitectura. Dado que los mockups son comprensibles por los usuarios finales, los mismos son utilizados como punto de partida para especificar conceptos de software utilizando modelos de alto nivel, fragmentando el flujo de trabajo lineal clásico de las metodologías MDWE en pequeños ciclos no lineales en los que los usuarios finales puedan participar activamente y ver la aplicación en funcionamiento. Como se verá luego, esta misma estrategia es aplicable a contextos MDD en general. Finalmente, en lugar de descartar los mockups una vez construidos (como se hace en la mayoría de los enfoques Ágiles [12–14]), los mismos son transformados en especificaciones de UI independientes de la plataforma, y enriquecidos de forma incremental, lo cual permite construir principalmente modelos de dominio, navegación y presentación (entre otros) de un modo iterativo a partir de ellos. De esta manera, los mockups sirven como una herramienta de elicitación de requerimientos y, a su vez, como punto de partida para el proceso de desarrollo de la aplicación.

El principal proceso resultante – llamado *Mockup-Driven Development Web* o, en su forma corta, *MockupDD Web* – es MDWE, pero evita las dependencias en cascada de las propuestas clásicas en este campo las cuales hacen que el proceso de obtención de prototipos funcionales de la aplicación en construcción sea lento. La construcción de mockups iniciales es el único paso obligatorio para comenzar el proceso de modelado. Sin embargo, debido a que los mockups frecuentemente se utilizan también como un artefacto requerimientos, esto no representa una sobrecarga adicional en el proceso – debido a que los mockups serán construidos de uno u otro modo. Una vez definidos los mockups, los requerimientos capturados con los usuarios finales se pueden modelar de forma rápida directamente sobre los mismos, reduciendo el esfuerzo de traducir artefactos de requerimentos (en este caso, principalmente mockups) en artefactos de software. Como se muestra en este trabajo, este proceso encaja bien en el contexto de metodologías ágiles como Scrum [17] y, al mismo tiempo, hace uso de metodologías MDWE conocidas y probadas en lugar de definir otra propuesta MDWE diferente.

Dado que el espectro de características que pueden ser modelados en Aplicaciones Web es extremadamente amplio, la parte de este trabajo orientada al modelado Web centra su enfoque en Aplicaciones Web data-intensive (es decir, *cuya finalidad principal es la presentación y manipulación de gran cantidad de datos* [1], como la metodología WebML [18]). En adición, también se muestra cómo puede utilizarse la misma estrategia de modelado para la definición de APIs RESTful, también en el contexto de desarrollos Web. Al mismo tiempo, algunos aspectos no funcionales como la facilidad de uso y calidad de presentación se pueden evaluar directamente mediante mockups a lo largo del ciclo de modelización.

En este trabajo de tesis se describirá una técnica de modelado centrado en mockups, introduciéndola primero con MockupDD Web. Luego de haber ejemplificado el desarrollo centrado en mockups para metodologías MDWE, se describirá un marco general para el modelado sobre mockups (lo que se referenciará como metodología MockupDD *general*) y una instanciación particular de esta metodología además de la ya presentada para la Web. Las principales contribuciones de la metodología MockupDD y sus especializaciones son:

(1) Mejorar el workflow *en cascada* de las metodologías MDD tradicionales (en especial, aquellas MDWE) para permitir acortar las iteraciones y permitir a usuarios finales y clientes involucrados interactuar más rápidamente son el software generado, agilizando el proceso.

(2) Mejorar la productividad en el modelado de aplicaciones a través de la técnica de modelado de aplicaciones sobre mockups reduciendo sus errores y tiempo requerido, como se mostrará luego en las secciones de validación

(3) Introducir a clientes y usuarios finales en el proceso de desarrollo, utilizando artefactos de especificación de requerimientos de fácil comprensión para los mismos en lugar de conceptos de modelado comprensibles sólo para desarrolladores. En este contexto se utilizarán mockups y diferentes técnicas de anotación y especificación de requerimientos formales sobre los mismos.

(4) La implementación de un lenguaje de modelado sobre mockups con diferentes *vistas*, satisfaciendo al mismo tiempo la necesidad de comprensión de requerimientos por parte de los usuarios finales y las capacidades técnicas requeridas por los desarrolladores.

(5) Con ayuda de esta versatilidad del lenguaje y del uso de mockups (los cuales son comprensibles en su totalidad por usuarios finales), proveer un método de modelado más trazable desde el punto de vista de los requerimientos.

2. Contexto en el que se realiza la propuesta de tesis

En esta sección se discutirán diferentes tópicos, problemáticas y prácticas comunes en lo relativo a procesos de desarrollo de software modernos y que dan origen a la definición de MockupDD como metodología. En primer lugar se discutirá el uso de prototipado y modelado de interfaces de usuario en el desarrollo de aplicaciones. Luego, se abordarán dos tipos diferentes de metodologías de desarrollo: Model-Driven y Ágiles. Finalmente, se convergerá entre los tres tópicos discutidos (prototipado de UI y los dos tipos de metodologías abordadas) a fin de describir cómo MockupDD intenta combinarlos, proponiendo un nuevo proceso de desarrollo.

2.1. Prototipado y modelado de interfaces de usuario

La práctica de construir mockups para prototipar interfaces de usuarios y otros aspectos de las aplicaciones se ha vuelto cada vez más frecuente en los procesos de desarrollos modernos. Un indicador de esta creciente popularidad es la gran cantidad de herramientas que han aparecido para crear mockups como por ejemplo Balsamiq[1], Pencil[2], Mockingbird[3], Moqups[4], OverSite[5], por sólo nombrar algunas. El principal objetivo de los mockups es ayudar a discutir las especificaciones de UI con los usuarios finales, así como también descubrir y definir los requerimientos asociados en un lenguaje que les es totalmente comprensible. Esto los diferencia de las especificaciones de texto estructurado convencionales, las cuales suelen contener jerga de negocios y términos ambiguos [7,15]. En adición, los mockups de UI funcionan no sólo como artefacto de captura requerimientos, sino también como un mecanismo de asistencia general en el proceso de elicitación[19]. Ricca et al. han realizado estudios estadísticos a partir de los cuales muestran cómo los mockups ayudan a mejorar la captura de requerimientos en comparación con los métodos tradicionales basados en texto, sin que ello implique una sobrecarga de esfuerzo adicional en el proceso [8]. Además, los mockups han sido propuestos como una herramienta exitosa para capturar y registrar requerimientos *fluidos* [20] – aquellos que por lo general se expresan oralmente o informalmente y son una parte implícita (generalmente, perdida) del proceso de elicitación. La importancia de la captura de requerimientos asociadas a los prototipos de interfaz de usuario (algo que se aborda en detalle en este trabajo) también se describe en el trabajo de Ravid et al. [7].

En lo relativo al prototipado, existen tres niveles genererales de detalle:

- Alta Fidelidad (Hi-Fi): Están orientados a la construcción de interfaces de usuario visualmente completas y que pueden ser potencialmente usables. Dentro de este conjunto pueden encontrarse desde dibujos detallados hasta UIs implementadas utilizando código en un determinado lenguaje de programación.

[1] Balsamiq Mockups – http://balsamiq.com/products/mockups
[2] Pencil Project – http://pencil.evolus.vn
[3] Mockingbird – https://gomockingbird.com
[4] Moqups – http://moqups.com
[5] OverSite – http://taubler.com/oversite

- Media Fidelidad (Mid-Fi): Dan importancia al contenido de la interfaz, pero dejando de lado aspectos de presentación detallados como tipografía, colores, etc. Ejemplos de estos prototipos son aquellos construidos con herramientas como Balsamiq o Pencil.
- Baja Fidelidad (Lo-Fi): Se orientan a capturar información general para obtener una visión general de lo que se espera de la UI, sin entrar en detalles visuales refinados. Dentro de este tipo de prototipos pueden encontrarse aquellos dibujados en lápiz y papel, en pizarrones o utilizando notas del estilo *post-it*.

El prototipado y modelado de interfaces de usuario es un campo ampliamente estudiado. Una gran cantidad de metodologías y lenguajes de modelado de UI han sido definidos [21,22]. En general, los mismos permiten un modelado de interfaces en uno o más niveles de abstracción claramente separados. No obstante, existen herramientas como DENIM [23] las cuales muestran una variación interesante, ofreciendo prototipar interfaces en diferentes niveles conceptuales y dibujando *a mano alzada*, sin disponer de un modelo formal como requerimiento para el modelado. SketchiXML [24] es otra herramienta similar la cual no sólo permite la prototipación de UI a mano alzada sino que también provee asistencia en tiempo real al diseñador con el objeto de detectar patrones de UI y, a través de los mismos, agilizar el modelado. Sin embargo, todas las herramientas de prototipado, desde las más sencillas hasta las más complejas como DENIM o SketchiXML, están centradas en la creación de bocetos de UI que son desechados luego de la etapa de elicitación de requerimientos – a no ser que se desarrolle la UI siguiendo el estricto proceso de modelado de interfaces que cada una propone.

Por otro lado, las técnicas de modelado de UI están usualmente orientadas a la definición de la interfaces de usuario en modo *top-down* (es decir, de más abstracto a más concreto). Una de los lenguajes más reconocidos en este contexto y bajo esta estrategia es UsiXML [25]. Este lenguaje promueve el desarrollo de UIs de modo multidireccional, en 4 niveles de abstracción particulares y varios contextos de uso diferentes. Los 4 niveles de abstracción que define son:

- Tareas y Conceptos (T&C): Describe las tareas a realizar a través de la UI y los conceptos de dominio involucrados en cada una.
- UI Abstracta (AUI): Define los espacios de interacción o presentación, agrupando las tareas del nivel anterior por ciertos criterios.
- UI Concreta (CUI): Refina la AUI definiendo objetos de interacción concretos como widgets, layouts, navegación etc., de un modo independiente de la plataforma.
- UI Final (FUI): Es la UI operacional – es decir, corriente en una plataforma concreta.

Aunque este tipo de lenguajes y metodologías permiten modelar la UI de forma ordenada y precisa, varios de los niveles de abstracción utilizados no son fácilmente comprensibles por usuarios finales – en particular, aquellos similares a T&C y AUI. Mientras que el uso de estos modelos puede ser fructífero para los desarrolladores, los mismos no son de fácil comprensión para usuarios finales y, por lo tanto, no permiten integrarlos de forma temprana en el proceso de modelado de la UI.

2.2. Metodologías Model-Driven para Web Engineering

Las metodologías de Desarrollo Dirigido por Modelos (Model-Driven Development o MDD) surgen como una solución a la desconexión entre la etapa de modelado y de implementación presentes las metodologías de desarrollo basadas en código. En los procesos de desarrollo habituales, la etapa de modelado en general tiene como resultado un conjunto de modelos para abstraer aspectos de implementación que son complejos o muchas veces desconocidos en los momentos tempranos del desarrollo. Luego, en la etapa de implementación subsiguiente, los modelos son interpretados por los programadores, quienes manualmente convierten las descripciones abstractas en implementaciones concretas. Esta traducción manual e informal es propensa a errores humanos frecuentes en el desarrollo, los cuales resultan en una disminución de la calidad del software y en la productividad alcanzada durante su construcción.

La separación entre las etapas de modelado e implementación implica también mantener la consistencia entre los modelos y sus implementaciones: siendo manual la tarea de conversión entre ambos, ocurre que cualquier cambio en alguna de ellas implica una modificación en su contraparte, la cual deberá ser una tarea también manual y, por lo tanto, propensa a errores. Dado que el costo de mantener los modelos actualizados es mayor que el beneficio inicial obtenido por los mismos, usualmente las actualizaciones no son efectuadas al cambiar aspectos en la implementación, lo cual a los torna inservibles como documentación.

En la Fig. 1 se describen esquemáticamente diferentes maneras de relacionar el modelo con el código fuente y la aplicación final, incluyendo procesos MDD *puros*. Las alternativas más usuales en la actualidad son:

(a) No incluir ningún modelo, lo cual es razonable para software de tamaño reducido.

(b) Se incluyen modelos que luego son descartados, debido a que el costo de mantenerlos actualizados es mayor a los beneficios que aportan. Este caso representa el uso tradicional de mockups en procesos de desarrollo ágiles.

(c) Se utilizan modelos para visualizar el código mediante alguna técnica de ingeniería inversa, lo cual puede facilitar su comprensión, pero no agrega semántica alguna al mismo.

(d) Ida y vuelta entre modelos y código, intentando actualizar los cambios producidos en cualquiera de los dos en su contraparte. Esto es posible de manera automática sólo cuando los formatos son estructuralmente parecidos o cuando se actualizan los aspectos comunes entre ambas partes.

(e) Model-Driven puro: los modelos permiten derivar la totalidad del código fuente de la aplicación.

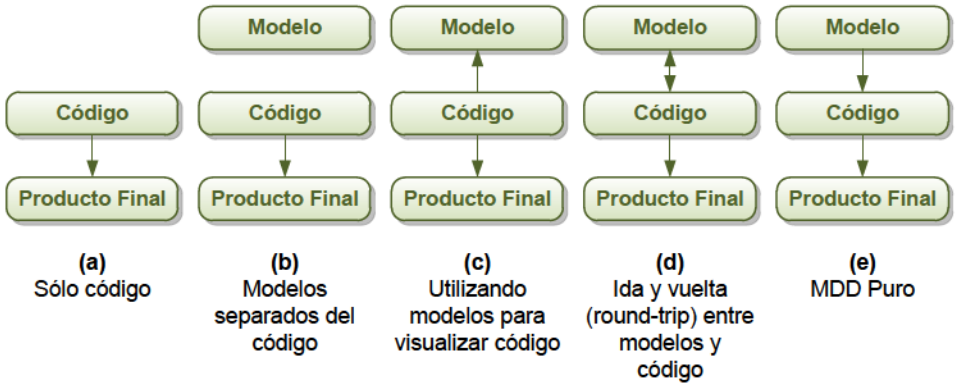


Figura 1. Diferentes estrategias para relacionar modelos y el código fuente de una aplicación.

Los enfoques MDD puros, a diferencia de los casos anteriores, tienen a los modelos como la fuente de información fundamental para derivar el código, obteniéndose luego la implementación final a partir de este último. Al generarse el código fuente desde los modelos, los problemas anteriormente comentados a causa de la separación entre las etapas de modelado e implementación no se presentan.

En la actualidad, los ampliamente usados lenguajes de Tercera Generación permiten, a través de compiladores, generar código de bajo nivel confiable el cual no tiene que ser modificado por los desarrolladores. Esto muestra que la utilización de lenguajes de mayor nivel de abstracción no sólo es factible, sino que además conlleva a un incremento considerable en la productividad. Esta misma factibilidad y demostración de éxito es la que motiva al desarrollo de soluciones MDD que provean una abstracción similar a la que los lenguajes de Tercera Generación proveen en comparación al lenguaje ensamblador, pero en dominios más específicos [26].

Toda solución MDD se caracteriza por:

- Tener un dominio acotado, basando su lenguaje en un conjunto de conceptos, relaciones y reglas provenientes del dominio del problema. Cuando más acotado sea el dominio del lenguaje, mayor será la semántica aportada de sus modelos.
- Alto nivel de abstracción, el cual permite describir la solución de software en términos del dominio, usando el lenguaje definido.
- Lenguaje (metamodelo), el cual especifica reglas sintácticas que deberán respetar los modelos así como también definen aspectos semánticos. Tanto reglas como semántica son cercanos al dominio y son representados por elementos utilizados en los modelos y sus relaciones.
- Generación de código, a través de la cual se transforman los conceptos de alto nivel expresados en los modelos a soluciones corrientes de software.
- Framework de dominio, el cual consiste en una o más piezas de software las cuales funcionan como una interfaz entre el código generado y la plataforma subyacente. El objetivo de este framework es, en la medida de lo posible, elevar el nivel de abstracción nativo de la plataforma destino para facilitar la generación de código.

Las metodologías de Model-Driven Web Engineering (MDWE) representan un subconjunto de las metodologías MDD cuyo dominio específico es el desarrollo de Aplicaciones Web. Por este motivo, sus

metamodelos incluyen características presentes en el ámbito Web tales como de páginas, links, comportamientos client-side, definición de servicios, entre muchos otros. Las primeras metodologías MDWE surgieron a principios de la década de 1990 y fueron evolucionando y mejorando en base a los rápidos cambios tecnológicos de la Web y a las metodologías de desarrollo en boga. En la Fig. 2 puede observarse un resumen cronológico de la evolución de las metodologías MDWE más populares, extraído de [5]. Metodologías MDWE como WebML [18], UWE [2] o OOHDM [3] tienen una larga trayectoria proponiendo mejoras en el campo del desarrollo Web. El enfoque de modelado utilizando conceptos de alto nivel y derivación automática de código evita los errores humanos involucrados en la codificación manual y también permite a los desarrolladores focalizarse sobre los aspectos semánticamente relevantes de la Aplicación Web en desarrollo en lugar de lidiar con las problemáticas frecuentes asociadas a la escritura de código.

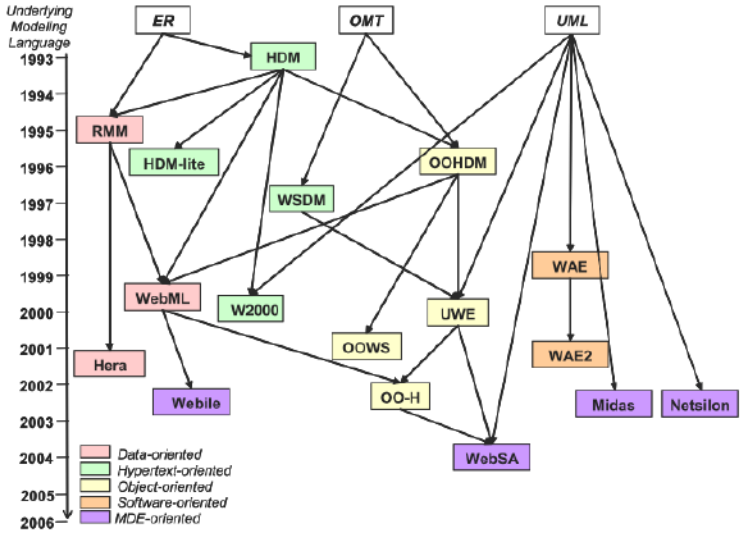


Figura 2. Evolución de las diferentes metodologías MDWE.

En general, las metodologías MDWE llevan a cabo su modelado atravesando diferetes concerns o *capas*, refinando especificaciones hasta alcanzar una descripción completa la cual puede utilizarse para generar el código de la Aplicación Web final. Las etapas generales que todas las metodologías generalmente definen son [5]:

- Modelado de Contenido: Expresa los objetos de dominio y sus propiedades sobre los cuales la Aplicación Web es definida.
- Modelado de Hipertexto: Incluye los requerimientos relativos a la navegación y nodos navegacionales. También especifica cómo se mostrará la información cuya estructura fue definida con anterioridad en los modelos de contenido.
- Modelado de Manejo de Contenido: Especifica las actualizaciones y manipulación de los objetos del dominio y sus relaciones en el contexto de nodos navegacionales.
- Modelado de Presentación: Refina las especificaciones de Hipertexto para definir los aspectos visuales de grano fino – por ejemplo, layout, colores, tipografías, etc.

Como puede deducirse de esta secuencia de etapas, los procesos MDWE están estructurados en forma lineal (en *cascada*), donde para modelar determinado concepto es necesario haber modelado conceptos dependientes en los modelos anteriores. Por ejemplo, no es posible definir un nodo navegacional que mostrará una lista de Fotografías sin ante haber definido el concepto de Fotografía y su estructura.

2.3. Metodologías Ágiles

Las Metodologías Ágiles (MA) definen procesos de desarrollo de software iterativo, en donde el foco principal es incluir a las partes interesadas en la construcción del producto (usuarios finales, gerentes, etc., normalmente llamados *stakeholders*) intensivamente en el proceso de desarrollo. El objetivo principal de estas metodologías es orientar el desarrollo a la entrega de software del mayor valor agregado posible para los stakeholders en iteraciones cortas, con el fin de fomentar su validación constante, adaptarse mejor a cambios imprevistos y reducir los riesgos durante el proceso. Los fundamentos principales de las MA se encuentran definidos en el Agile Manifesto[6], el cual promueve hacer énfasis en:

- Individuos e interacciones en lugar de en procesos y herramientas,
- Software funcional en lugar de documentación extensiva,
- Colaboración de los clientes en lugar de negociación de contratos y
- Rápida respuesta frente a cambios en lugar de seguir un plan detallado

De acuerdo con encuestas y análisis recientes, el 84% de las organizaciones consultadas utilizaron algún tipo de MA en sus proyectos [10] durante el año 2013, lo cual muestra el gran grado de aceptación que han cobrado estas metodologías y sus principios en los procesos de desarrollo.

Existen varios procesos que se definen como *ágiles* e implementan sus principios, cada uno con ciertas variaciones o particularidades. De todos ellos, el proceso Scrum (en el que se basa parte de este trabajo, y que se detallará luego) es el más usado, abarcando un 55% de uso en la industria y combinado en algunos casos con otros procesos.

Las MA están basadas fuertemente en la escritura de código fuente como base de la especificación e implementación de software. Se utilizan técnicas como testing e Integración Continua [27] para asegurar que el producto construido a través de la codificación manual sea correcto. En lo relativo a captura de requerimientos, en general se utilizan artefactos de rápida construcción y que sean lo más precisos posibles en relación a lo que los usuarios finales esperan. Dos de estos artefactos extensamente usados en este contexto son mockups y User Stories [12]. En lo relativo a diseño e implementación, en las procesos ágiles también suelen construirse modelos, aunque estos son utilizados como una especificación rápida y luego desechados en etapas posteriores del desarrollo [28].

Las MA comparten ciertos conceptos y prácticas con las técnicas de Diseño Centrado en el Usuario (User-Centered Design o UCD) [29]. Estas tienen como objetivo incorporar la perspectiva del usuario en el proceso de desarrollo de software haciendo énfasis en obtener un sistema usable desde el comienzo del mismo [30]. Un elemento clave en estos procesos son, entre otros, las actividades de prototipado [31]. Los

[6] Manifesto for Agile Software Development – http://agilemanifesto.org/

enfoques de prototipado han sido propuestos como una solución principal para manejar y/o inclusive estimar cambios de diseño durante el desarrollo de sistemas. Los métodos de prototipado en este contexto van de Lo-a Mid-Fi [32]: bocetos en papel, storyboards, simulaciones guionadas, etc.

Uno de los artefactos de requerimientos más utilizando en los contextos ágiles son las User Stories [16]. Las US describen características que son valorables desde el punto de vista de los usuarios de la aplicación y se caracterizan por focalizarse en cierta funcionalidad puntual en el contexto de cierto rol de usuario, siendo muy acotadas en longitud. Esto las hace especialmente atractivas en el contexto de las MA, en las cuales se intenta evitar el uso de documentación extensa, reemplazarla por comunicación fluida entre desarrolladores y stakeholders y obtener software funcional lo más rápido posible para que sea testeado por los usuarios finales. Las US están compuestas por 3 aspectos fundamentales:

- Una descripción de la Story, utilizada para planificar y como recordatorio de lo que debe satisfacerse haciendo énfasis en la funcionalidad valorable para el usuario.
- Conversaciones acerca de la Story, las cuales sirven para ampliar sus detalles.
- Tests que sirvan para acordar y documentar detalles los cuales puedan ser utilizados para determinar cuando una US se ha satisfecho por completo.

En general, un patrón clásico para la escritura de US es *Como [rol de usuario], deseo/debo poder [descripción de la acción o funcionalidad a realizar]*. En general el tamaño de una US en términos de funcionalidad debe ser tal que permita su codificación y testeo en, como máximo, dos semanas. Las US más grandes son llamadas *Epics* y en general son fragmentadas en dos o más sub-Stories las cuales puedan ser atacadas dentro de los plazos comentados con anterioridad. Una práctica habitual es escribir las US en tarjetas (llamadas *Cards*) las cuales representan los requerimientos de los usuarios. Estas Cards también son utilizadas para anotar detalles de las Conversaciones que se tienen con los usuarios – a fin de que los detalles de las mismas no se pierdan.

2.4. La problemática de integrar MA y MDD

Tanto las MA como las metodologías de MDD intentan reducir el esfuerzo del desarrollo de software. Las primeras buscan hacerlo recurriendo a la codificación manual y a técnicas y estrategias que permitan integrar más fácilmente a clientes o usuarios finales en el desarrollo. El uso de artefactos de requerimientos centrados en el cliente, la preparación para cambios recurrentes y constantes y las iteraciones cortas son sólo algunas de las técnicas que utilizan las MA para integrarlos. Para compensar los errores humanos producto de la codificación directa, estas metodologías plantean la utilización de técnicas que agregan una carga de trabajo extra al proceso de desarrollo, como la escritura de tests y la utilización de entornos de Integración Continua.

Las metodologías MDD proponen una estrategia diferente: elevar el nivel de abstracción para generar software de un modo más productivo. Utilizando modelos de alto nivel y generadores de código se asegura la calidad del software modelado. Sin embargo, ciertos aspectos positivos relacionados con los procesos ágiles no pueden ser utilizados en el contexto de sus procesos, debido a cómo están estructuradas sus etapas de

modelado. En el ámbito Web, en el caso particular de las MDWE, lograr un producto funcional para los stakeholders implica la construcción lineal de una secuencia de modelos hasta alcanzar una Aplicación Web funcional. Los artefactos de elicitación de requerimientos, a pesar de que en muchos casos pueden llegar a tener cierta formalidad (como mockups o Casos de Uso) son descartados en las etapas siguientes del desarrollo, al igual que en las MA. Finalmente, en este contexto la mayoría de los modelos contienen conceptos que no son fáciles de comprender para usuarios finales, lo cual hace aún más complicada la tarea de integrarlos al proceso de construcción de la aplicación.

En este trabajo se propondrá MockupDD como una estrategia para integrar MA y MDD, permitiendo:

- La utilización de User Stories como elementos básicos y ágiles de captura de requerimientos valorables desde el punto de vista de los usuarios finales.
- La construcción temprana de mockups asociados a las US como artefacto fundamental de requerimientos, favoreciendo no sólo el uso de especificaciones visuales concretas de la aplicación, sino también utilizándolos como un lenguaje que tanto desarrolladores como stakeholders pueden comprender sin ambigüedades.
- La reutilización de estos mockups, interpretándolos e importándolos como modelos de UI sencillos – dejando de lado las complejidades de metodologías de modelado de interfaces de usuario como UsiXML.
- Modelar ágil e iterativamente aspectos de negocios e interacción sobre los mockups previamente, sin tener interdependencia de modelos que retarden el proceso.
- Fomentar que la mayor parte posible de la funcionalidad sea modelada sobre los mockups que fueron comprendidos y validados por stakeholders, favoreciendo la trazabilidad de las especificaciones.
- Utilización de un lenguaje de modelado con una complejidad reducida y/o con diferentes sintaxis de un modo tal que sea técnicamente viable para el equipo de desarrollo y, a su vez, comprensible por usuarios finales.

En la sección siguiente se introducirá la metodología MockupDD especializada para la Web (MockupDD Web), a fin de mostrar cómo la misma se puede aplicar a un dominio el cual no sólo es muy popular, sino también para el cual ya se han provisto variadas metodologías dirigidas por modelos – varias de las cuales fueron comentadas en la Sección 2.1.

3. MockupDD Web: Integrando agilidad y modelado en el contexto de la Web

Uno de los objetivos principales de un proceso basado en mockups como MockupDD es comenzar el proceso de desarrollo utilizando modelos que los desarrolladores y los clientes pueden entender, con el fin de obtener feedback constante de estos últimos. Por lo tanto, el desarrollo en el contexto de MockupDD comienza con la construcción de mockups de UI sin comportamiento alguno. Por esta razón, el proceso se ha llamado Mockup-Driven Development o simplemente *MockupDD,* como se ha introducido hasta el momento. En particular, en esta sección se describirá MockupDD Web, la especialización de la metodología MockupDD orientada, al igual que WebML, desarrollo de Aplicaciones Web data-intensive utilizando modelos.

MockupDD Web acepta indistintamente mockups construidos con herramientas digitales (como el representado en la Fig. 3.a.), así como también en archivos HTML estáticos tradicionales - como el mockup HTML representado en la Fig. 3.b. Este soporte heterogéneo permite adaptar la propuesta a la técnica de mockuping que sea más amena a los desarrolladores. Para ilustrar el enfoque, se mostrarán principalmente mockups HTML, ya que están más cerca de lo que los usuarios finales ven cuando trabajan con la aplicación final y también porque están soportados de forma nativa en las herramientas de soporte que se describirán más adelante.

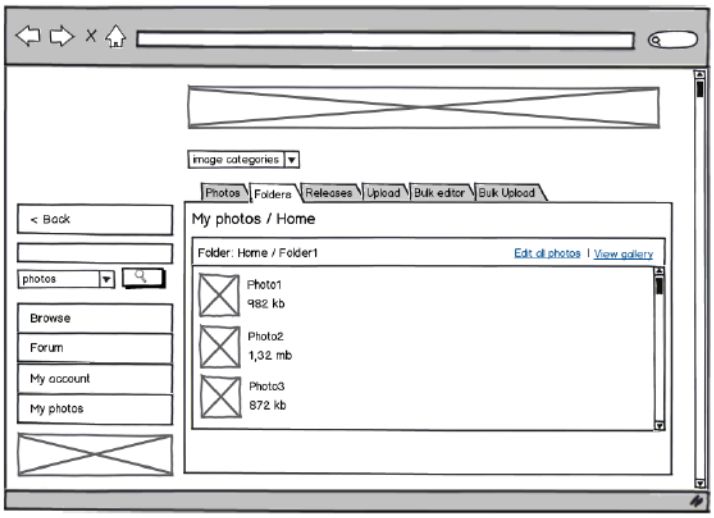
3.1. Visión general del proceso

El proceso MockupDD Web comienza con una rápida fase de elicitación de requerimientos la cual define un conjunto de User Stories a satisfacer. Los desarrolladores, con participación intensiva de clientes y usuarios finales, pasan inmediatamente a construir mockups para representar las User Stories gráficamente (Fig. 2, Paso 1). En esta etapa el equipo de desarrollo captura los principales conceptos en estos mockups en un Modelo Estructural de Interfaz de Usuario – llamado *Structural User Interface model* o SUI. En el mismo se identifican los componentes concretos de UI involucrados en las funcionalidades descriptas en las User Stories [16] y se asocian a su representación gráfica en los mockups. Todo elemento del SUI tendrá entonces un *widget* de interfaz de usuario relacionado que lo representa de modo gráfico en un mockup. El metamodelo del SUI, su motivación y las diferentes estrategias para su obtención desde mockups se detallarán en las siguientes secciones.

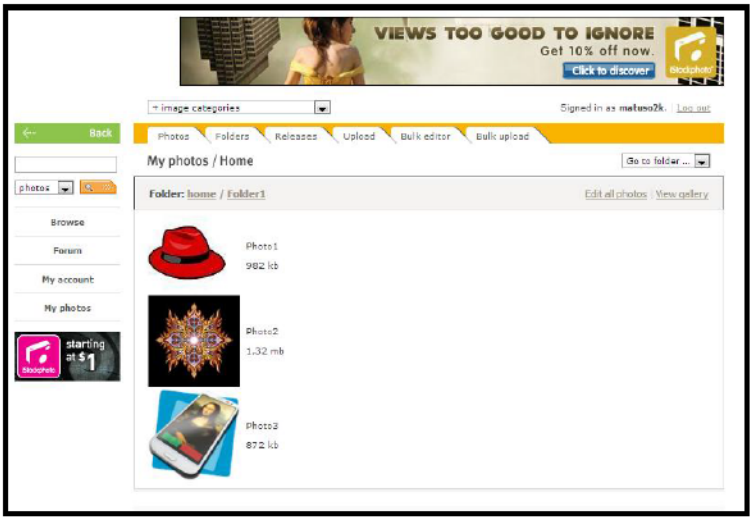
Una vez que todos las US fueron representadas con mockups y se definieron sus conceptos de modelo SUI (Fig. 2, Paso 2), los requerimientos asociados a interacción, comportamiento, modelo conceptual y lógica de negocios se traducen en especificaciones utilizando anotaciones simples – las cuales son llamadas *tags*. En esta etapa, el equipo de desarrollo introduce estos tags sobre los widgets en los mockups (previamente importados como elementos del modelo SUI) interpretando la semántica subyacente en los mismos y en las User Stories y plasmándola como especificaciones de requerimientos y de comportamiento/interacción al mismo tiempo. En esta etapa es esencial la interacción con los usuarios finales para resolver cualquier potencial ambigüedad. Dado que el modelo SUI conserva la referencia a los

widgets originales en los mockups, aunque los tags parecieran ser aplicados sobre los *widgets* gráficos en los mockups estos son, de hecho, aplicados sobre los elementos del SUI. De este modo, el modelo SUI almacena no sólo la estructura de los elementos de UI relevantes para la implementación de las User Stories, sino también las especificaciones que las implementan – en forma de tags. Esto permite:

1. Mantener una estrategia de enriquecimiento independiente de la herramienta de prototipado utilizada – ya que los modelos SUI son independientes de la plataforma, como se verá más adelante.
2. La aplicación de las especificaciones de un modo intuitivo utilizando los artefactos de requerimientos en su representación original (mockups), lo cual hace que las mismas sean claramente trazables y facilita la compresión de las especificaciones por parte de los clientes u usuarios finales.



(a) Mockup construido con la herramienta Balsamiq



(b) Mockup construido utilizando HTML

Figura 3. Ejemplo de mockup construido con diferentes enfoques.

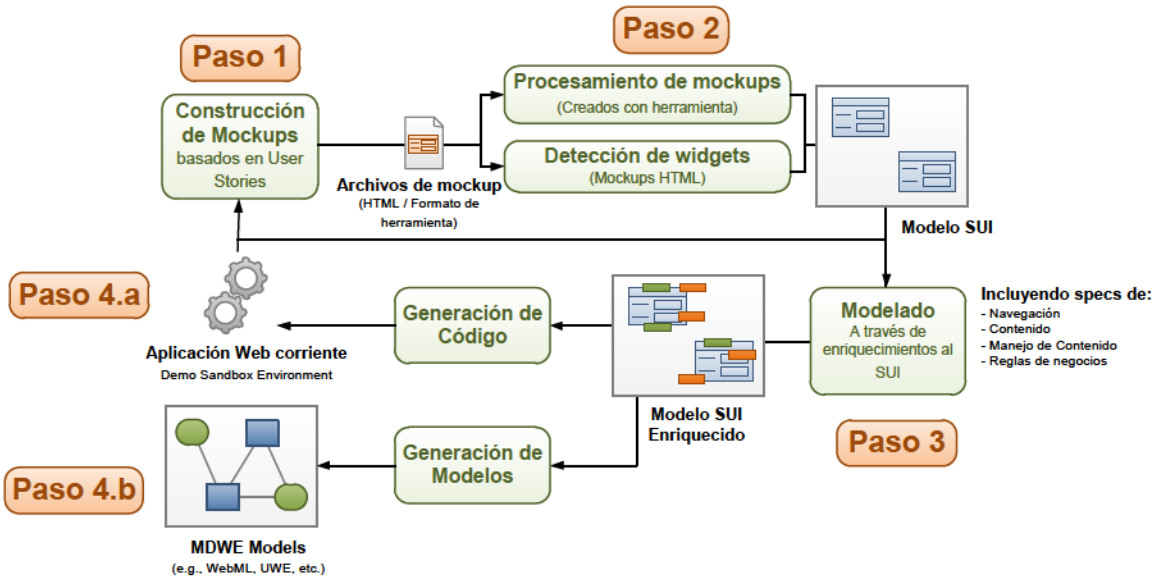


Figura 4. Workflow de una iteración MockupDD Web, incluyendo pasos técnicos.

En la Fig. 4 puede observarse el workflow de trabajo general de una iteración de MockupDD Web. En las siguientes subsecciones se mostrará cómo se utilizan los tags de forma no estructurada para ayudar a definir y refinar las especificaciones de forma iterativa, preservando la agilidad. El resultado de este paso (Fig. 4, Paso 3) es un modelo SUI completamente enriquecido – es decir, *taggeado*. En este punto, los desarrolladores pueden ejecutar inmediatamente una demo a través de la herramienta de soporte para comprobar si los requerimientos fueron capturados correctamente (Fig. 4, Paso 4.a). Esta demo simula cómo se comportará la aplicación final sin que la misma haya sido derivada para ser ejecutada en la plataforma destino. Todas las etapas del proceso, desde la construcción de los mockups hasta a la ejecución de una aplicación de demostración se pueden aplicar de forma incremental en presencia de los usuarios finales hasta alcanzar una implementación aceptable para la iteración actual, o hasta que se encuentren requerimientos que implican un mayor tiempo de ejecución. Si las nuevas funcionalidades suponen modificación alguna de los mockups, el proceso debe comenzar desde el Paso 1, creando nuevos mockups o refinando los existentes y enriqueciéndolos y para luego generar una nueva demostración de funcionamiento de la aplicación. Por otro lado, si los requerimientos recién descubiertos no requieren cambios en la estructura de interfaz de usuario, el proceso puede empezar desde el Paso 3, enriqueciendo mockups existentes hasta que se cumplan los nuevos requerimientos. Ambos casos implican una interacción y comunicación cruciales entre el equipo de desarrollo y los clientes o usuarios finales para asegurar que el producto resultante está funcionando como se espera.

Toda la iteración termina con la generación de modelos SUI taggeado y la posterior derivación de la aplicación final (Fig. 4, Paso 4.b). Más cerca del final de la iteración, las especificaciones pueden ser refinadas con el apoyo de las herramientas provistas o inclusive utilizando lenguajes específicos o codificación directa. Esto permitirá tomar decisiones de modelado y diseño finales por parte del equipo de

desarrollo. La Fig. 5 resume los actores involucrados en cada paso del proceso de MockupDD Web y sus responsabilidades.

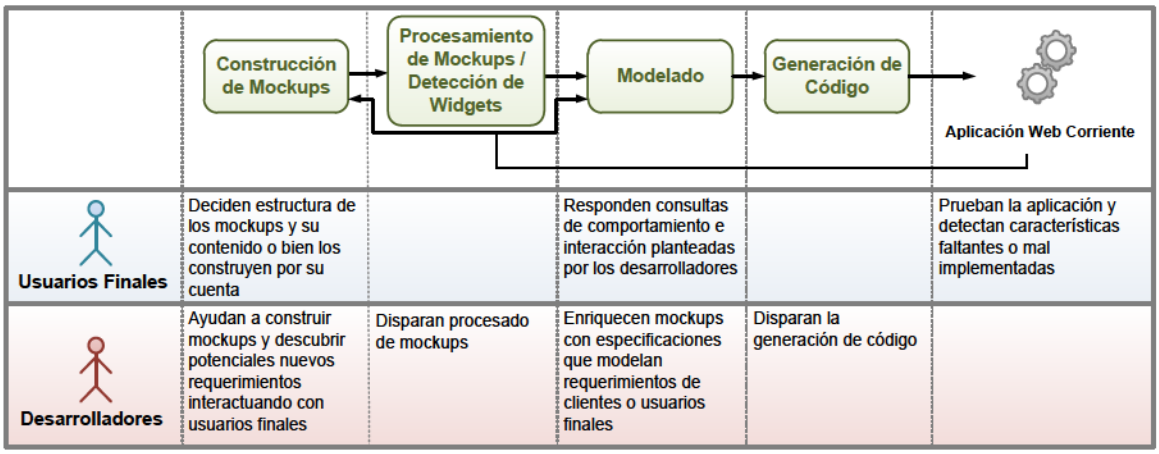


Figura 5. Actores involucrados en las diferentes etapas del proceso MockupDD Web.

3.2. Aplicación de ejemplo

Para ejemplificar cada etapa en la metodología, se utilizará un sitio Web de ejemplo que se llamará *Photo Stock*. El mismo se ha utilizado como caso de estudio y también para llevar a cabo la validación de la metodología descripta más adelante. El sitio Photo Stock consiste en una Aplicación Web convencional de manejo de fotografías de stock. A través de la misma, sus usuarios pueden subir fotografías, categorizarlas, organizarlas y también cobrar por permisos de uso para para terceros. Asimismo, la aplicación facilita un blog personal para cada usuario, el cual es utilizado en general para publicar contenidos relacionados con su producción fotográfica. Finalmente, también se proporcionan foros de discusión para sus usuarios.

Algunas de las User Stories de esta aplicación son:

- **User Story 1.** Como Usuario, quiero poder crear, eliminar y cambiar el nombre de las carpetas en las cuales voy a almacenar mis fotografías.
- **User Story 2.** Como Usuario, deseo subir nuevas fotos una carpeta existente.
- **User Story 3.** Como usuario, deseo poder publicar un nuevo post en mi blog personal.

En las siguientes secciones se utilizará el ejemplo del sitio Photo Stock para mostrar cómo se ejecutan los diferentes pasos de la metodología en el contexto del desarrollo de una Aplicación Web concreta, satisfaciendo algunas de estas User Stories.

3.3. Paso 1 y Paso 2: Construcción y procesamiento de mockups

Como se comentó con anterioridad, el workflow de MockupDD Web comienza por elicitar requerimientos utilizando User Stories y mockups. El número de mockups necesarios depende de cómo los

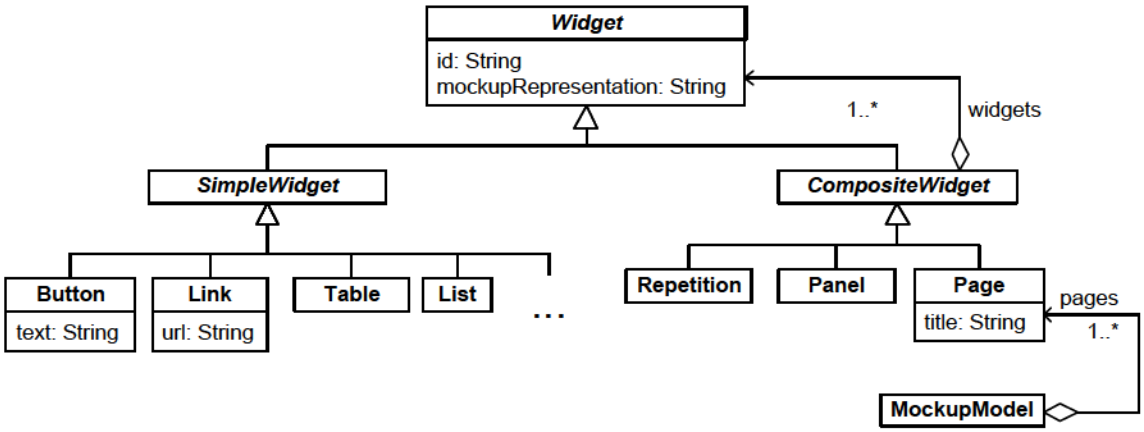
pasos de interacción estén distribuidos en diferentes pantallas dentro de la aplicación. Adicionalmente, varias User Stories pueden compartir uno o más mockups, ya que pueden estar relacionadas a funcionalidades que tengan puntos en común.

Para formalizar la estructura de la interfaz de usuario, se utiliza el metamodelo SUI [33]. El mismo representa la estructura de una interfaz de usuario mínima necesaria para especificar características de interacción y comportamiento relacionadas con elementos visuales. La estructura principal del metamodelo se puede observar en la Fig. 6.a. El metamodelo SUI es muy similar a otros metamodelos y lenguajes de UI como UsiXML [25], XAML[7] o XUL[8], y define una clase abstracta `Widget` (que representa cualquier elemento visual) y los subclasifica en elementos simples – `SimpleWidgets` – y compuestos – `CompositeWidgets`. Todo `Widget` posee una propiedad `id` que lo identifica y un string `mockupRepresentation` el cual se puede utilizar para asociarlo a su representación de origen en un mockup (de ser identificable) en pos de proveer trazabilidad. A su vez, el metamodelo SUI agrupa los `Widgets` en unidades de interacción (llamadas `Screens`) las cuales, en el contexto de MockupDD Web, serán representadas por páginas HTML. Ejemplos de `SimpleWidgets` son, por ejemplo, `Button` (que representa un botón clickeable en la UI) o `Inputbox` (que representa un cuadro para ingreso de texto). El conjunto de `CompositeWidgets` admitido en el modelo SUI son, en adición a las mismas `Screens`, los `Panels` y `Repetitions`. Mientras que los primeros agrupan widgets con cierta funcionalidad en común, los segundos instancian una cierta cantidad de copias de sus `Widgets` internos de acuerdo a cierto criterio – por ejemplo, para mostrar una lista de elementos con sus propiedades internas.

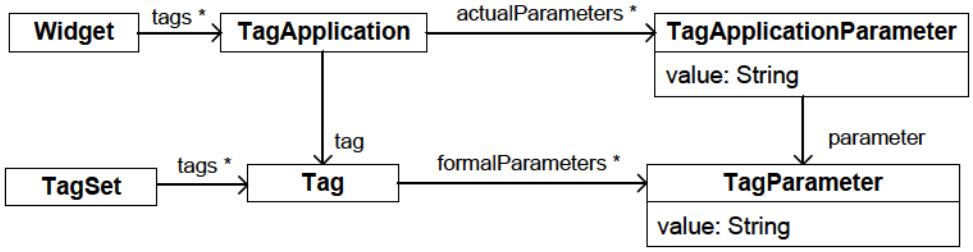
En lugar de considerar sólo la definición de una estructura de interfaz de usuario detallada (como UsiXML, XAML o XUL), el metamodelo SUI está diseñado para soportar la especificación de un conjunto ampliable y personalizable de características sobre elementos de interfaz de usuario a través de *tags,* que serán tratados en detalle en la sección siguiente.

[7] XAML Overview (WPF) – http://msdn.microsoft.com/en-us/library/ms752059.aspx

[8] XUL | Mozilla Developer Network – https://developer.mozilla.org/en/docs/XUL



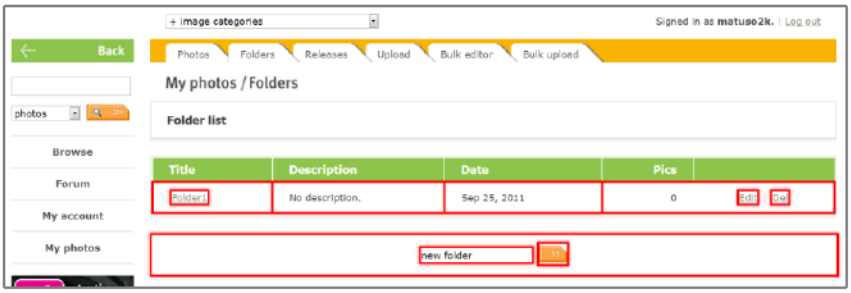
(a) Estructura principal del metamodelo SUI



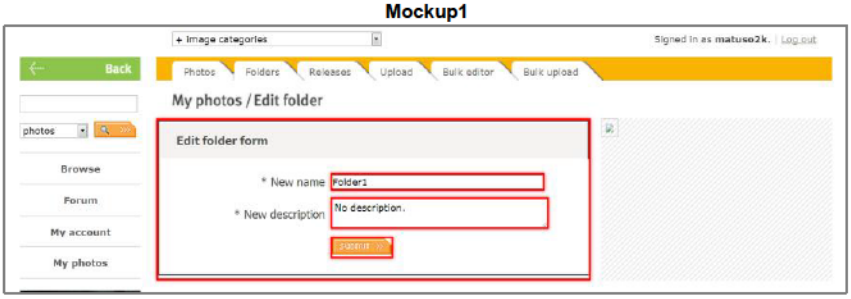
(b) Metamodelo de tags sobre SUI

Figura 6. Diferentes partes del metamodelo SUI.

En la Fig. 7 se observan dos mockups HTML construidos en relación a la User Story 1 de la aplicación Photo Stock. Un SUI model fue extraído para cada uno luego de su creación, referenciando las partes principales de la interfaz de usuario – es decir, aquellos widgets que son importantes desde el punto de vista del comportamiento necesario para satisfacer la User Story. En la figura, las partes resaltadas de cada página no son parte de los mockups en sí, sino una decoración visual agregada por la herramienta de MockupDD Web. Estas partes resaltadas denotan elementos SUI capturados a partir de elementos de la UI – es decir, componentes de las páginas que tienen un elemento SUI asociado. Una vez seleccionados y asociados todos los widgets involucrados en la User Story, se avanza hacia la etapa de *tagging*.



Mockup1



Mockup2

Figura 7. Mockups construidos para la User Story 1.

En la Fig. 6.a se describen sólo algunos de los `SimpleWidgets` que incluye el SUI. La lista de `SimpleWidgets` más relevantes considerados por el SUI comprende:

- `Button`: Representa un botón clickeable.
- `Link`: Representa un link clickeable – similar a un `Button`, pero con diferente aspecto.
- `DropdownList`: Representa una lista desplegable de elementos, la cual permite seleccionar sólo uno – el cual es mostrado por defecto.
- `Inputbox`: Representa un cuadro en el cual puede ingresarse texto.
- `Checkbox`: Representa un cuadro que puede marcarse o desmarcarse para indicar la activación o desactivación de cierta propiedad o la asignación de determinado valor booleano a un objeto.
- `List`: Representa una lista de elementos en donde cada uno es representado textualmente – por ejemplo, una lista de usuarios en el que cada uno es representado por su nombre.

3.4. Paso 3: Tagging

Enriquecer los mockups (en esta etapa, SUI models) con diferentes tipos de especificaciones es una parte clave de la metodología, dado que representa la estrategia de modelado ágil e iterativo que la caracteriza. Como se dijo antes, este modelado es llevado a cabo utilizando *tags*. Un tag es una especificación atómica compuesta por un nombre y cero o más parámetros textuales (siguiendo la forma sintáctica, `Tag(Param1, Param2, ..., paramN)`. Cada tipo de tag puede aplicarse sólo sobre un subconjunto de widgets SUI específico. En adición, cada uno puede definir una sintaxis particular para cada uno de sus parámetros, extendiendo su semántica tanto como sea necesario. Los tags se agrupan en *tag sets* para aislar *concerns* específicos (por ejemplo, navegación o manipulación de datos) que pueden abordarse por separado en la mayoría de los casos. A través de esta separación, se evitan las dependencias secuenciales entre los diferentes

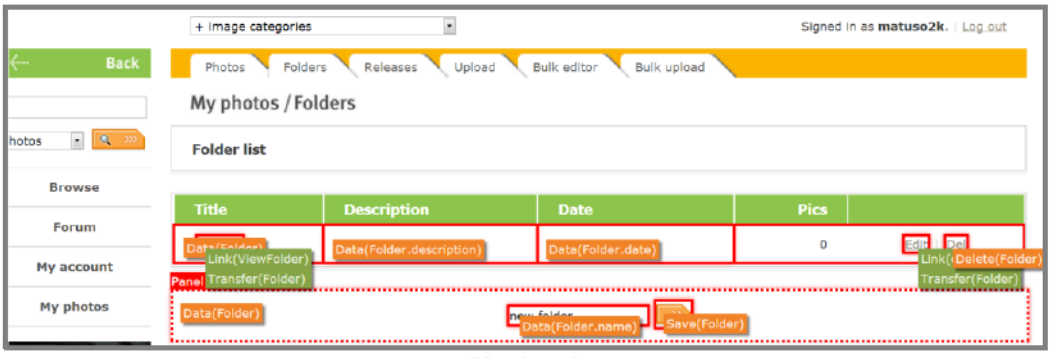
concerns durante el modelado tradicional MDWE, las cuales atentan contra la rápida obtención de prototipos funcionales y, por lo tanto, restringen la agilidad. Un ejemplo de este este enfoque de modelado puede ser la modelización de funciones de navegación sin que sea necesario un modelo de contenido anterior. Los tags tienen una función doble: por un lado, se usan para plasmar en términos formales, concretos y ejecutables los requerimientos de los usuarios finales; por otro lado, funcionan (al igual que los mockups) como un artefacto de comunicación en sí mismo entre clientes y/o usuarios finales y desarrolladores, proveyendo una sintaxis técnica pero a su vez simple en la mayoría de los casos. Por ejemplo, las etiquetas `Data(Folder)` y `Save Folder)` – tratadas en detalle más adelante – pueden aplicarse cuando un usuario final o cliente expresa que un mockup particular debe crear una instancia de un `Folder` usando los campos de entrada provistos en el mismo. Un `Folder` es un objeto de negocios que identificado por los usuarios finales (utilizando su propia *jerga*) y, al mismo tiempo, a través de su formalización en el tag, pasa a ser una especificación de software valorable – por ejemplo, una clase `Folder` en un modelo de clases o una entidad en un modelo Entidad-Relación. De la misma manera, si otro usuario final establece que inmediatamente después de crear un `Folder` la aplicación debe mostrar una listas actualizada con todos los `Folders` de la aplicación, este requerimiento se puede implementar agregando un tag `Link(folders)` sobre un botón del mockup existente. Estas dos especificaciones (la primera, relacionada con el modelo de contenido de la aplicación y la segunda, asociada a la navegación) se pueden definir de forma independiente, en cualquier orden e incluso como resultado de requerimientos especificados y descubiertos por diferentes usuarios finales, clientes y/o desarrolladores.

En la Fig. 6.b se representa cómo los tags se estructuran formalmente, agrupan y se aplican *sobre* widgets SUI desde la perspectiva metamodelo MockupDD. Como muestra la figura, los tags (representados por la clase `Tag`) son agrupados en tag sets (clase `TagSet`) y pueden contener una lista de cero o más parámetros (`TagParameter`). Cero o más `Tags` pueden aplicarse sobre un mismo `Widget`, aplicación la cual es representada por la clase `TagApplication`. Un `Widget` tiene una lista de `TagApplication` (la cual representa todos los tags aplicados al mismo) y cada una hace simplemente hace referencia al tag aplicado y a los valores reales correspondientes a sus parámetros necesarios – una lista de `TagApplicationParameter`.

Los requerimientos detallados expresados en la User Story 1 de la aplicación Photo Stock se pueden implementar con la especificación de los siguientes tags (ver Fig. 8):

- El tag `Data(Folder)` sobre el `Panel` en el Mockup2 indica que el mismo editará o creará instancias de `Folder`.
- Los tags `Data(Folder.name)` y `Data(Folder.description)`, aplicados sobre los `Inputboxess` en Mockup2, especifican que los mismos permitirán establecer (y obtener) los atributos `nombre` y `description` de un `Folder`.
- El tag `Save(Folder)`, aplicado sobre el botón *Submit* del Mockup2, indica que la acción predeterminada cuando se hace clic en el mismo consistirá el almacenar una instancia de la clase `Folder` que se está editando en su panel contenedor.

- Los tags `Data(Folder)`, `Data(Folder.name)` y `Save(Folder)` aplicados sobre el `Panel` inferior en el Mockup1 tienen la misma semántica que los tags idénticos y anteriormente comentados, implementando una funcionalidad de creación rápida de `Folders`.
- El tag `Data(Folder)` en la `Repetition` superior (Mockup1) especifica que el mismo mostrará una lista de `Folders`. En adición, la descripción de un `Folder` será mostrada en el `Label` interno taggeado con `Data(Folder.description)`, así como también su fecha de creación – a través del tag `Data(Folder.date)`.
- El tag `Delete(Folder)` aplicado sobre el `Button` *Del* indica que un click en él producirá la eliminación del `Folder` referenciado en su `Widget` contenedor (en este caso, una fila de la tabla de `Folders`)
- Los tags `Link(EditFolder)` – cubierto visualmente por el tag `Delete()` previamente comentado – y `Transfer(Folder)` aplicados sobre el `Button` *Edit* especifican que un click en el mismo producirá una navegación a la pantalla EditFolder (Mockup2), transfiriendo el `Folder` referenciado en su panel contenedor como parámetro. Debido a que el Mockup2 permite la creación y almacenamiento de instancias de `Folder`, si se pasa un `Folder` como parámetro la misma UI puede ser utilizada para la edición del mismo.



Mockup1

Mockup2

Figura 8. Mockups de la Fig. 7 con tags MockupDD Web aplicados los cuales describen el comportamiento esperado.

Los tags generales que brinda MockupDD Web son los siguientes:

- **Node(<nodeId>)**
  - **Semántica**: Indica que la página actual será identificada con el nombre <nodeId>.
  - **Tag Set**: Navegación/Interacción
  - **Aplicable sobre**: Screen
- **Link(<nodeId>)**
  - **Semántica**: Un click o una interacción similar sobre el Widget al que se aplica este tag disparará una acción de navegación hacia la página (Screen) identificada como <nodeId>.
  - **Tag Set**: Navegación/Interacción
  - **Aplicable Sobre**: Button/Link
- **Data([<widgetId>:]<typeRef>)**
  - **Semántica**: El widget sobre el cual se aplica mostrará o permitirá editar una o más instancias de <typeRef>. Opcionalmente, se podrá especificar un widget sobre el cual se obtendrá la instancia a mostrar o editar (utilizando la construcción <widgetId>:). La sintaxis de <typeRef> (descripta más adelante) puede ser más compleja que el nombre de una clase (por ejemplo, describiendo una propiedad y su clase asociada, siguiendo el patrón <clase>.<propiedad>).
  - **Tag Set**: Contenido
  - **Aplicable Sobre**: Widget
- **Select()**
  - **Semántica**: El widget sobre el cual se aplica será utilizado para seleccionar objetos a fin de realizar determinada acción a posteriori.
  - **Tag Set**: Navegación/Interacción
  - **Aplicable Sobre**: SimpleWidget
- **Transfer(<typeRef>$_1$, <typeRef>$_2$, ..., <typeRef>$_N$)**
  - **Semántica**: Cuando el widget sobre el cual se aplica este tag dispare una acción de navegación, se transferirán las instancias de datos especificadas por **<typeRef>$_1$, <typeRef>$_2$, ..., <typeRef>$_N$**.
  - **Tag Set**: Navegación/Interacción
  - **Aplicable Sobre**: Button/Link
- **Save(<typeRef>$_1$, <typeRef>$_2$, ..., <typeRef>$_N$)**
  - **Semántica**: Un click o interacción similar sobre el widget sobre el cual se aplica este tag provocrá que las instancias de datos especificadas por **<typeRef>$_1$, <typeRef>$_2$, ..., <typeRef>$_N$** sean persistidas.
  - **Tag Set**: Contenido
  - **Aplicable Sobre**: Button/Link
- **Delete(<typeRef>$_1$, <typeRef>$_2$, ..., <typeRef>$_N$)**

- **Semántica**: Un click o interacción similar sobre el widget sobre el cual se aplica este tag provocrá que las instancias de datos especificadas por **<typeRef>$_1$, <typeRef>$_2$, ..., <typeRef>$_N$** sean eliminadas.
- **Tag Set**: Contenido
- **Aplicable Sobre**: `Button/Link`

- **`Associate(<typeRef>`$_1$`, <typeRef>`$_2$`[, <associationName>])`**
  - **Semántica**: Un click o interacción similar sobre el widget sobre el cual se aplica este tag provocrá que las instancias de datos especificadas por **<typeRef>$_1$** y **<typeRef>$_2$** sean asociadas. Opcionalmente se puede especificar el nombre de la relación específica con `<associationName>`; en caso de que esta propiedad sea omitida los objetos se asociarán asumiendo un nombre de asociación genérico.
  - **Tag Set**: Contenido
  - **Aplicable Sobre**: `Button/Link`
- **`Dissociate(<typeRef>`$_1$`, <typeRef>`$_2$`[, <associationName>])`**
  - **Semántica**: Un click o interacción similar sobre el widget sobre el cual se aplica este tag provocará que las instancias de datos especificadas por **<typeRef>$_1$** y **<typeRef>$_2$** sean disociadas. Opcionalmente se puede especificar el nombre de la relación específica con la cual estos dos objetos fueron anteriormente disociados con `<associationName>`; en caso de que esta propiedad sea omitida se asumirá el nombre de asociación por defecto.
  - **Tag Set**: Contenido
  - **Aplicable Sobre**: `Button/Link`
- **`Query(<queryDescription>, <typeName>)`**
  - **Semántica**: El `Panel` sobre el cual se aplica este tag mostrará una o más instancias de `<typeName>`. Los criterios por el cual se obtendrán las instancias a ser mostradas serán formalmente descriptos por el parámetro `<queryDescription>`. El lenguaje formal utilizado para describir estos criterios es dependiente de la plataforma – por ejemplo, puede usarse SQL. Pueden referenciarse el valor de otros widgets como parámetro describiéndolos con el patrón `:<widgetId>` dentro del contenido de `<queryDescription>`.
  - **Tag Set**: Contenido
  - **Aplicable Sobre**: `Panel`
- **`Action(<description>,`** **<typeRef>$_1$, <typeRef>$_2$, ..., <typeRef>$_N$)**
  - **Semántica**: Un click o interacción similar sobre el widget sobre el cual se aplica este tag provocrá que se ejecute una acción arbitraria, descripta por el parámetro `<description>`, pasándose como parámetro a esta acción as instancias de datos especificadas por `<typeRef>`$_1$, `<typeRef>`$_2$, ..., `<typeRef>`$_N$. Al igual que en el caso del tag

Query(), el lenguaje o sintaxis que se utilizará para describir la acción será dependiente de la plataforma.

- **Tag Set**: Contenido
- **Aplicable Sobre**: Button/Link

Mientras que los tags (y sus concerns implícitos) se pueden aplicar en cualquier orden, un tag puede especificar más de una característica en la aplicación siendo modelada. Por ejemplo, el tag Data(Folder) no sólo implica que el contenido de un Panel permite ver o modificar una instancia de Folder, sino que también especifica la existencia de una clase Folder en el mismo. De la misma manera, los tags Data(Folder.description) y Data(Folder.name) no sólo asocian widgets de interfaz de usuario a propiedades específicas de un objeto de clase Folder sino que también declaran su existencia en le modelo de dominio.

Como se comentó en su descripción formal, los tags Action() y Query() permiten utilizar cualquier sintaxis o lenguaje dependiente de la plataforma para describir acciones o consultas de datos. Estos tags facilitan la introducción de aspectos que no pueden abstraerse en MockupDD Web. Aunque los mismos atentan contra la abstracción – dado que son dependientes de la plataforma – le permiten a los desarrolladores introducir especificaciones que exceden a la semántica disponible en los tag sets estándar siguiendo las prácticas de modelado sobre mockups de MockupDD.

3.4.1. Tags de datos

Los tags de datos – aquellos que tienen la forma Data([widgetId:]<typeRef>) – tienen una sintaxis propia la cual permite definir variados elementos correspondientes al modelo de datos de la aplicación siendo modelada. Esto facilita una construcción iterativa de dichos modelos de un modo trazable – dado que todo elemento del modelo de datos tiene, al menos desde el punto de vista del usuario, su metáfora asociada en un mockup. Las diferentes construcciones que permite <typeRef> son:

- **[*]Class**. Denota que un objeto de clase Class es mostrado o permite ser manipulado a través de un elemento compuesto subyacente en la interfaz de usuario – es decir, un CompositeWidget SUI. Si se utiliza el * opcional – por ejemplo, en Data(*Post) –, entonces el tag especifica que se mostrará o permitirá manipular no una instancia sino una lista de instancias de clase Class.
- **Class.attribute[:datatype]**. Especifica que el atributo attribute del objeto Class es mostrado o permite ser editado a través de un widget concreto de la interfaz de usuario – un SimpleWidget SUI. Opcionalmente, se puede proveer un tipo de datos al atributo, el cual puede ser uno de Date, String, Double, Decimal, Boolean o Blob. Si no se especifica un tipo de datos concreto, se asume String.
- **Class1.association -> [*]Class2**. Denota que un objeto de clase Class2 es mostrado o puede ser manipulado a través de un elemento compuesto (CompositeWidget) subyacente en la interfaz de usuario. En adición, el elemento de clase Class2 es obtenido a través de otro

elemento existente de clase `Class1`, siguiendo la asociación nombrada `association` que los relaciona. Si se utiliza el modificador opcional *, se denota que se mostrarán o se permitirá manipular una lista de objetos `Class2` y, a su vez, se especifica que la cardinalidad de la asociación `association` de `Class1` a `Class2` es *uno a muchos*.

- **`Subclass => Superclass`**. Denota que un objeto de clase `Subclass` es mostrado o permite ser manipulado en la interfaz de usuario y que, a su vez, la clase a la cual pertenece (`Subclass`) hereda de la clase `Superclass`.

Las especificaciones atómicas que permite el tag `Data([widgetId:]<typeRef>)` pueden ser combinadas para, en una sola aplicación del tag, describir varios artefactos de modelo de datos y, a su vez, especificar cómo se obtienen las instancias de datos a mostrar y manipular a través de navegaciones. Por ejemplo, el tag:

```
Data(Post => Publication.comments -> *Comment => Annotation)
```

establece que la clase `Post`, que hereda de `Publication`, tiene una lista de `Comments` (la cual referencia a través de la asociación `comments`). A su vez, `Comment` es un tipo de `Annotation`. Finalmente, la asociación comments entre `Post` y `Publication` es heredada – es decir, es definida en `Publication`. Como puede verse en el ejemplo, con una sola anotación se lograron definir 7 elementos correspondientes al modelo de datos: 4 clases, 2 relaciones de herencia y una asociación. La evaluación de la eficiencia y eficacia de modelado los tags de datos utilizando esta sintaxis particular será abordada más adelante en este trabajo de tesis.

3.5. Refinamientos al Tagging

Los tags intentan ser tan conceptuales como sea posible con el fin de añadir agilidad al proceso de modelado, promoviendo una estrategia de modelado ágil *just barely good enough* [28]. En este sentido, estos permiten una especificación incremental a través del perfeccionamiento de sus parámetros o la combinación de su semántica cuando se encuentra utilizada en el mismo widget o relacionado en el mismo mockup. Se define como un *tag ambiguo* a aquel que no es semánticamente completo en sí mismo y que puede tener más de una interpretación semántica válida. Mientras que los tags de este tipo son más rápidos de aplicar, los mismos no tienen valor expresivo concreto hasta que son *refinados*. En algunos casos, este refinamiento puede hacerse automáticamente utilizando heurísticas, por ejemplo:

- El tag `Data()`, en su forma más básica puede escribirse como `Data(<class>)`. Sin embargo, como ya se ha comentado antes, este tag también permite la forma `Data(<widgetId>:<class>)` para especificar el widget origen que proporcionará la instancia de datos de tipo `<class>`. Si una `Repetition` (llamada `repetition1`) y un `Panel` (denominado `panel1`) se encontraran en la misma `Screen` y ambos tuvieran aplicados el tag `Data(Folder)`, es altamente probable que este último muestre los datos seleccionados de la primera y por lo tanto, el tag aplicado al `Panel` puede ser refinado a `Data(repetition1:Folder)`.

- Si dos diferentes `Screens` tienen el tag `Data(<class>)` aplicado sobre uno de sus widgets internos y una navegación se expresa de una a la otra utilizando el tag `Link()`, un tag `Transfer(<class>)` puede ser añadido automáticamente dado que esta navegación probablemente implicará una transferencia de un objeto asociado en la mayoría de los casos. Esto podría haber sido aplicado en Mockup1 (Fig. 8), si el tag `Transfer(Folder)` no hubiera sido especificado.

3.6. Generación de modelos

Tal como se definieron hasta el momento, los tags pueden ser considerados como especificaciones aisladas y atómicas con una semántica específica. En esta sección se muestra cómo los mismos pueden ser traducidos a elementos de modelos MDWE y combinados para especificar las características de diseño más complejas.

Como se mencionó anteriormente y se muestra en la Fig. 4, cada iteración en el proceso MockupDD Web implica agregar, cambiar o quitar tags. Después de que los tags han sido completamente definidos, generadores de modelos o código pueden utilizarse para obtener los modelos MDWE correspondientes. Se han definido diferentes generadores de modelos para las metodologías UWE [34] y WebML [6], dos de las metodologías MDWE más populares y actualizadas. Esto permite afirmar que (1) el metamodelo de tags de MockupDD Web puede ser semánticamente equivalente a metamodelos MDWE modernos y (2) al generarse modelos para estas metodologías desde mockups taggeados, pueden aprovecharse las características de generación de código que las mismas proveen. De este modo, MockupDD Web no pretende proveer otra solución MDWE más, sino funcionar como un enfoque de agilización a las existentes.

En la Fig. 8 se muestra cómo diferentes tags aislados pueden utilizarse para derivar conceptos MDWE atómicos en WebML y UWE. Como se ha mencionado con antelación, un mismo tag puede definir conceptos MDWE en varios tipos de modelos – por ejemplo, una clase en modelo de contenido y una Query (UWE) o Query Unit (WebML) a nivel de hipertexto. Es importante aclarar que, durante el desarrollo de esta tesis, el lenguaje WebML ha sido paulatinamente reemplazado por IFML[9], su versión estandarizada por la OMG. En este trabajo se ha utilizado su sintaxis original dado que (1) los conceptos Web en ambas versiones son similares y (2) gran parte de los trabajos publicados de MockupDD Web y comentados aquí hacen referencia a esta sintaxis.

[9] IFML – The Interaction Flow Modeling Language – http://www.ifml.org

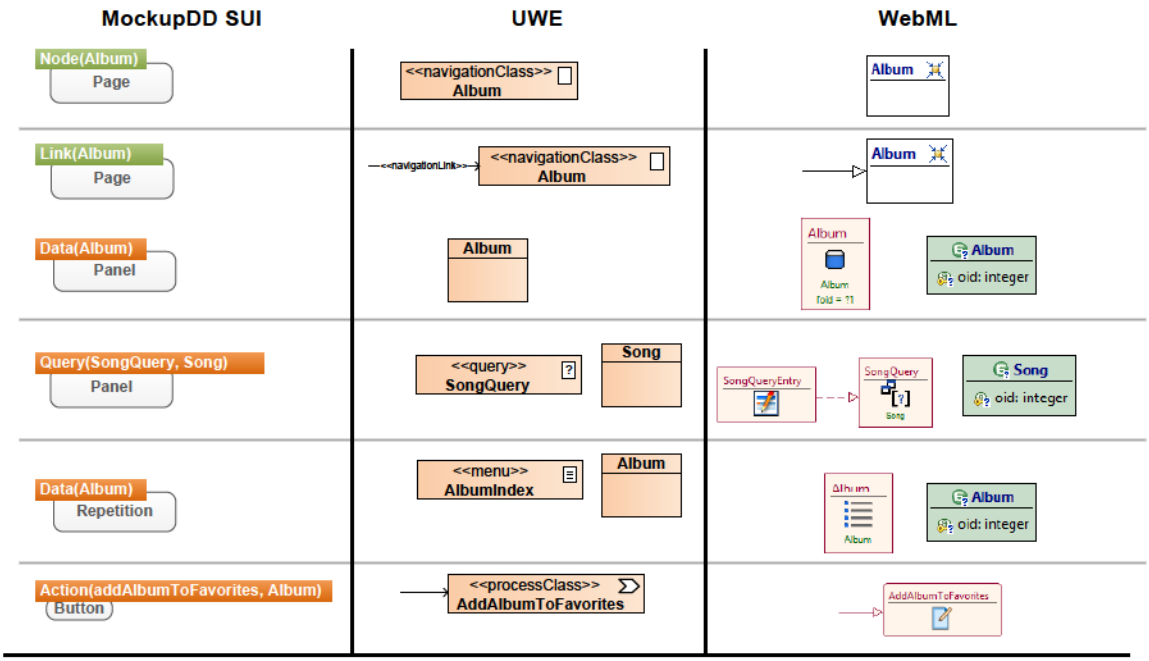


Figura 9. Semántica de los diferentes elementos SUI taggeados expresados en conceptos de UWE y WebML.

Luego de la etapa de tagging y una vez que los tags ambiguos han sido concretizados, se obtiene un modelo SUI completamente taggeado (de acuerdo a los requerimientos a satisfacer en esta iteración). Inmediatamente después, puede generarse una demostración corriente de la aplicación o bien un modelo MDWE para una metodología soportada. En ambos casos, el primer paso en el proceso de generación es derivar un modelo de especificaciones – conocido como *Spec Model*. Este tipo de modelo presenta una semántica más precisa y conceptual que ayuda a reducir el esfuerzo de traducción a los modelos MDWE. En el Spec Model, la representación textual interna de los tags (por ejemplo, los nombres de clases de los tag `Data()` o los nombres de las `Screens` identificadas en el tag `Link()`) se representan a través de un conjunto de *Spec Objects* relacionados. Esto permite que la parte de la generación de modelos MDWE que consiste en parsear el contenido textual y relacionar la semántica de los tags sea implementada una sola vez para todos los generadores de modelos y/o código. De este modo, la implementación de los generadores de modelos consistirá simplemente en iterar por el Spec Model y generar las representaciones MDWE correspondientes.

Por cada tag particular, existe al menos un tipo de Spec Object. Cada uno de estos puede tener otros Spec Objects asociados, así como también uno o más elementos del SUI model (comenzando por el `Widget` sobre el cual el tag que le dio origen fue aplicado, el cual debe ser referenciado obligatoriamente). El proceso de conversión de tags a Spec Objects consta de los siguientes pasos:

1. Por cada tag, se creará una instancia de Spec Object de acuerdo a su tipo concreto.
2. Por cada tag, se analizará la relación de contención en el modelo SUI para definir si se trata de un *hijo* de otro tag existente o no. Esto implicará recorrer, a partir del `Widget` directo sobre el cual fue aplicado el tag y de manera ascendente, a todos sus padres en la jerarquía de contención. Si se encuentra un widget SUI en esa jerarquía el cual ya tiene un tag aplicado, entonces el tag actual se considerará como *hijo* de éste último.

3. Por cada parámetro que permita referenciar a otro widget en el modelo SUI (por ejemplo, de tipo [widgetId:]<typeRef>) se buscará el widget con el id widgetId y se relacionará directamente.

En la Fig. 10 se muestra un ejemplo de cómo se obtiene un Spec Model para un SUI model dado. Por otro lado, la Fig. 11 muestra esquemáticamente cómo se lleva a cabo el procesamiento de tags partiendo desde su especificación potencialmente ambigua. En este contexto, el primer paso consiste en refinar los tags para quitar su ambigüedad (tal como se comentó anteriormente). Luego, se convierte cada tag en su Spec Object destino, de acuerdo a cómo se ha especificado este mapping en cada tag set. El mapping puede no ser 1 a 1 siempre, dado que la semántica de algunos tags puede ser diferente según su uso contextual – por ejemplo, un tag Data(<class>.<property>) puede ser utilizando para asignar valores a controles en la interfaz de usuario o bien para obtenerlos. En esta etapa, cada Spec Object es asociado con un elemento del SUI.

Finalmente, una vez obtenidos todos los Spec Objects que representen a los tags aplicados, se procede a asociarlos entre sí, utilizando la sintaxis precisa de los tags. Las asociaciones entre Spec Objects pueden darse de diferentes modos:

1. Para cada parámetro que haga referencia a otro widget del SUI en los tags precisos – por ejemplo, a través de la sintaxis <widgetId>:<typeRef> – se asociará su Spec Object al del otro tag aplicado sobre el widget con id <widgetId>.
2. Para tags que dependan de otros definidos en su widget contenedor – como el caso de los tags Data(<class>.<property>) dentro de widgets con tags Data(<class>) – se asociará el Spec Object actual al Spec Object aplicado a su tag contenedor.

Estos dos modos de asociación entre tags son utilizados por ciertos tipos de Spec Objects para asociarse con otros a fin de expresar determinada acción o interacción. Una vez completo y definido el Spec Model, haciendo uso de las funcionalidades de generación de modelos desarrolladas para MockupDD Web, se genera el modelo WebML representado en la Fig. 12 a partir de los mockups taggeados en la Fig. 8.

En la Fig. 13 se describe una parte del metamodelo de Spec Objects para los tag sets básicos que provee MockupDD Web. Como puede apreciarse en este metamodelo, todo Spec Object (clase SpecObject) tiene relacionado un Widget SUI. La subclase DataRelatedSpec agrupa Specs relacionadas con manejo de datos. Las clases SaveSpec y ReadSpec representan la semántica de almacenamiento y recuperación de clases de datos – es decir, tienen la misma semántica que los tags Save(<class>) y Data(<class>) respectivamente. Su atributo className (heredado de DataRelatedSpec) denota la clase sobre la cual están operando. La Spec ReadSpec en particular contiene las propiedades isList, query y navigationPath. La propiedad isList (booleana) indica si el Spec va a manejar una instancia o una lista de instancias de objetos de clase className. Por otro lado, query especifica una expresión de consulta a través de la cual se obtendrá el o los objetos a mostrar en la UI – modelando de este modo la semántica del tag Query(). Finalmente, una ReadSpec puede definir su propiedad navigationPath y especificar otra DataRelatedSpec como origen de datos (a través de la asociación dataSource). Esto define que se obtendrá una o más instancias desde su DataRelatedSpec relacionado y luego se navegarán una o más asociaciones a partir de estos (especificadas por la propiedad navigationPath) hasta alcanzar la o las

instancias a mostrar. Tanto `SaveSpec` como `ReadSpec` deben definir una lista de propiedades las cuales persistirán o leerán y mostrarán en la interfaz de usuario respectivamente; esto es representado por las clases `SavePropertySpec` y `ReadPropertySpec` respectivamente.

Cualquier acción a llevar a cabo en la interfaz de usuario es modelada como una lista de `SimpleActionSpecs`. Una de estas acciones son las `SaveSpec` anteriormente introducida, la cual implementa la persistencia de un objeto determinado. Otras dos `SimpleActionSpecs` son `AssociateSpec` y `DissociateSpec`, ambas subclases de `AssociationSpec`. Estas dos acciones referencian dos `DataRelatedSpecs` cuyos objetos serán relacionadas a través de una asociación particular, cuyo nombre es denotado en la propiedad `associationName`. Todas las acciones son agrupadas y listadas en orden en un objeto `ActionSpec`.

En la Fig. 14 se muestra otra fracción del metamodelo de Spec Objects, pero relacionado con la navegación. En este contexto, toda acción de navegación es representada por un `NavigationSpec` el cual es una subclase de `AtomicActionSpec`, como toda acción individual. Una `NavigationSpec` tiene el nombre de la página destino hacia la cual navegará (`destinationPage`) y puede transferir uno o más objetos de datos a esa página. Estas transferencias son denotadas por la clase `TransferSpec`, la cual posee un `DataRelatedSpec` que funciona como origen de datos para obtener el o los objetos a transferir. Un `TransferSpec` es una subclase de `DataRelatedSpec` dado que funciona también como un origen de datos para otros Spec Objects. Por ejemplo, al transferirse un objeto producto de una navegación, el `TransferSpec` que es utilizado en la página de origen para indicar la transferencia del objeto puede ser luego referenciado por un `DataReadSpec` como origen de datos en la página destino.

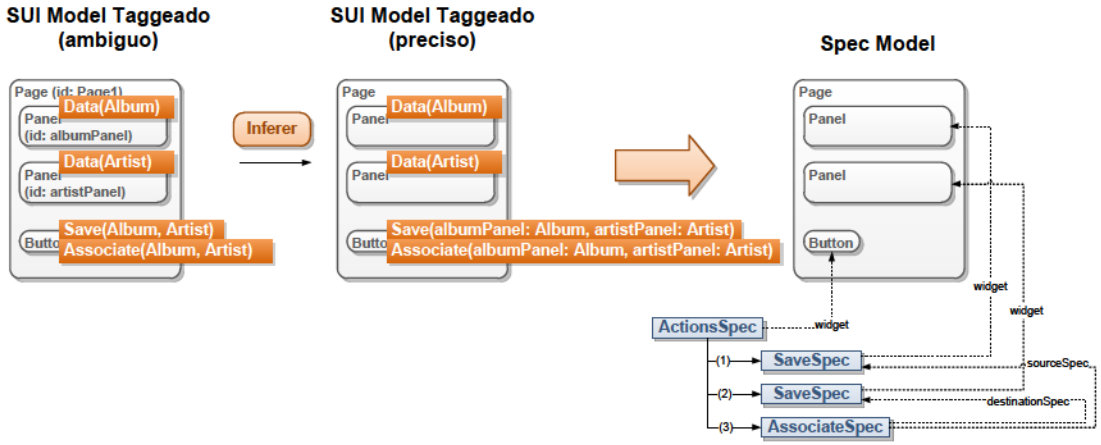


Figura 10. Obtención de un Spec Model desde un SUI con tags ambiguos.

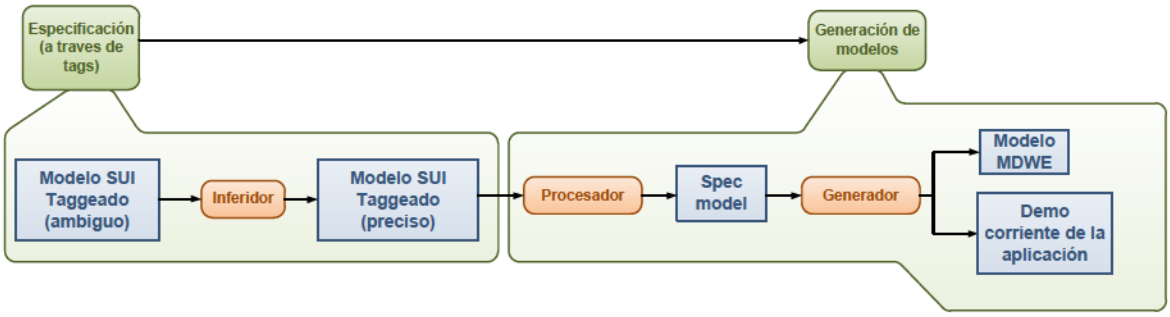


Figura 11. Proceso completo de tagging, incluyendo la generación del Spec Model.

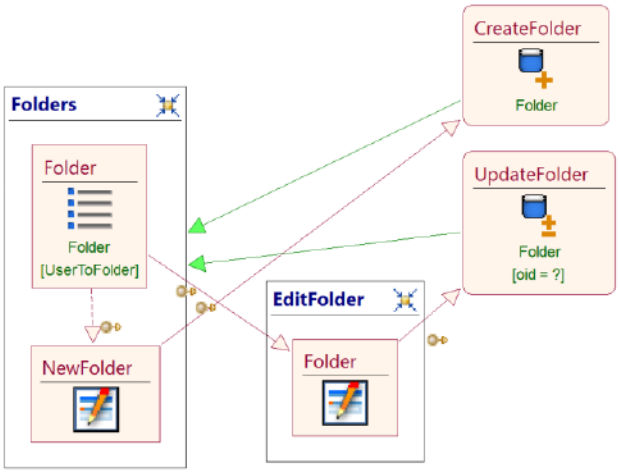


Figura 12. Modelo WebML generado a partir de los mockups taggeados de la Fig. 7.

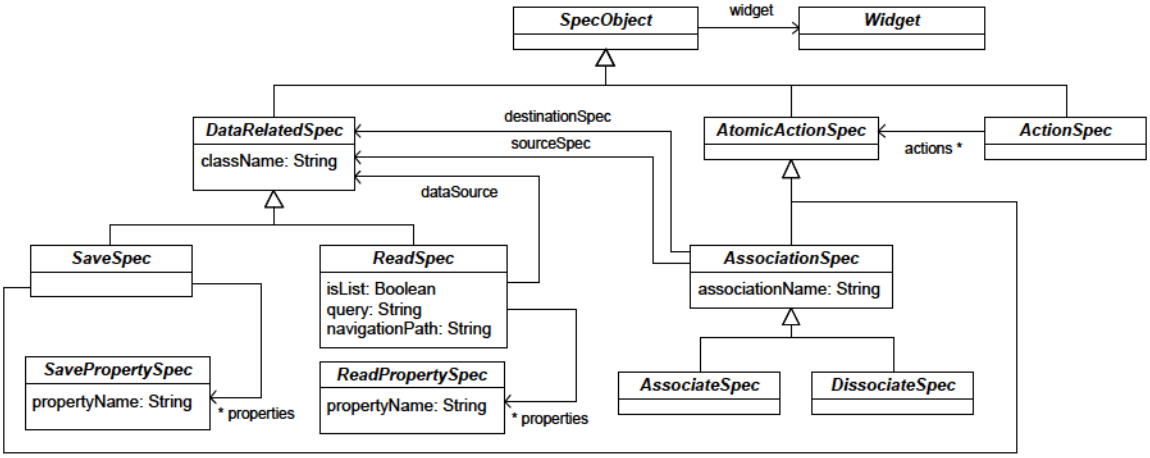


Figura 13. Metamodelo de Specs Objects (fracción de manipulación y definición de datos).

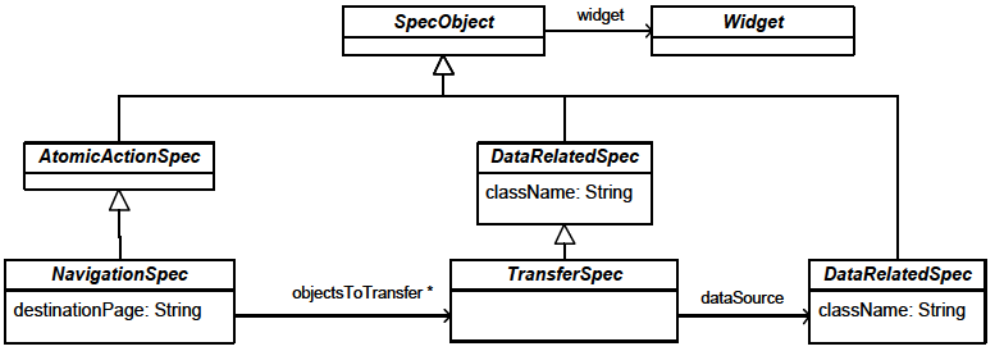


Figura 14. Metamodelo de Specs Objects (fracción de navegación).

Si bien en la Fig. 9 se mostró la relación general entre tags y elementos MDWE, en términos de semántica concreta, los elementos MDWE son generados directamente a partir de Spec Objects. Luego de procesados y obtenidos todos los Spec Objects como se comentó anteriormente, a cada `Screen` en el SUI se le asocia un árbol de Spec Objects que agregan semántica sobre sus `Widgets` internos y concretizan la semántica de los tags. El proceso de generación de código consiste en realizar uno o más recorridos sobre este árbol y generar, por cada Spec Object, cero o más elementos MDWE que sean necesarios. En general, se tendrá que realizar una cantidad de recorridos y generaciones parciales igual a la cantidad de modelos diferentes que requiera la metodología, en el orden que deban ser definidos. En la Fig. 15 se muestra el mapping entre diferentes versiones de `ReadSpecs` y sus elementos WebML a generar. En el caso de WebML, se requerirán al menos 2 etapas de generación: primero generar el modelo de dominio subyacente y luego el de hipertexto.

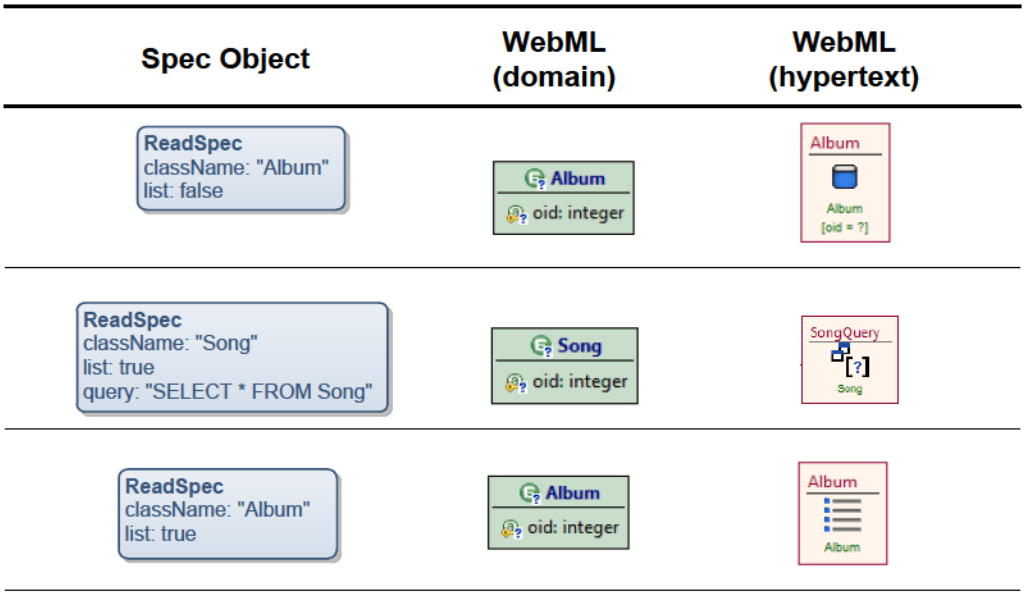


Figura 15. Mapping de Spec Objects a elementos WebML.

3.7. El Proceso de Desarrollo MockupDD

Hasta ahora, se han introducido los aspectos procedimentales y teóricos de MockupDD. En esta sección se introducirá brevemente el proceso de desarrollo de la metodología. El mismo es una adaptación del proceso de desarrollo ágil Scrum [17].

El proceso Scrum (véase la Fig. 16.a) comienza con la construcción de un *Product Backlog*, que es una lista de todas las características que el producto de software debe poseer, priorizados por valor entregado al cliente. El producto se construye de forma iterativa en *Sprint*s. Cada Sprint representa una iteración en el proceso de desarrollo y comienza con una reunión de planificación (*Sprint Meeting*) que tiene como fin la confección de un plan detallado. En este plan, un subconjunto de la lista de tareas remanentes en el Product Backlog son concretizadas en forma de tareas de ejecutables, formando lo que se conoce como un *Sprint Backlog*. Una vez que este Backlog ha sido cuidadosamente definido para que tome no más de un mes de trabajo, el equipo de desarrollo comienza a resolver todas las tareas definidas en el mismo. Al principio de cada día durante el Sprint se realiza una reunión corta (conocida como *Daily Scrum Meeting*) para compartir el progreso del trabajo y los nuevos problemas y desafíos que se encuentran durante el mismo. Por último, al final del Sprint se hace una demostración de una versión entregable de la aplicación siendo desarrollada al *Product Owner* (el interesado en el desarrollo del producto de software). Pasado este punto, se hace una re-priorización del Product Backlog de ser necesario y se define la próxima meta general a alcanzar en el siguiente Sprint.

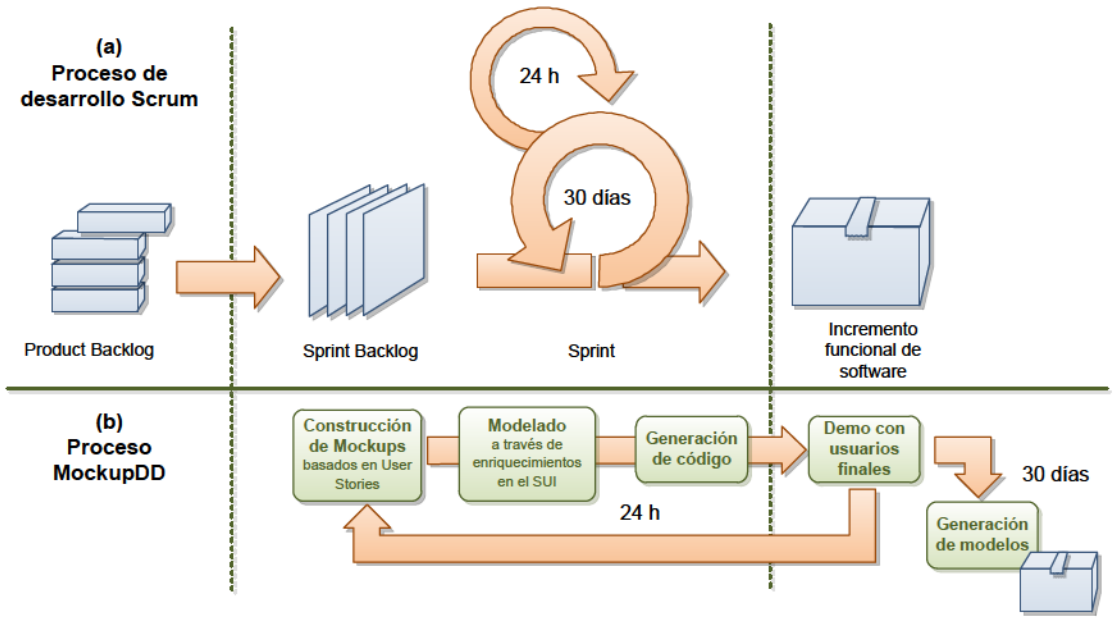


Figura 16. Workflow del proceso Scrum vs el MockupDD.

Con MockupDD (Fig. 16.b), el primer paso de un Sprint sigue la misma estrategia que Scrum *puro*, con la excepción de que el Sprint Backlog debe expresarse como una lista de User Stories cuyo nivel de detalle debe ser suficiente como para ser descripto utilizando los mockups. En lugar de utilizar codificación directa,

en cada Sprint se construyen mockups para concretizar las User Stories como se detalla en la Sección 3.1. Los mockups luego se convierten a una instancia del SUI y son taggeados. Luego de aplicar cierta cantidad mínima de tags, una demostración puede correrse con el objeto de mostrar a los usuarios finales o clientes cómo se comporta la aplicación modelada hasta el momento. De este modo, en lugar de esperar el final del Sprint, pequeños incrementos de la aplicación durante el mismo se pueden mostrar directamente a los stakeholders para que evalúen se comporta como se espera. Esto es posible gracias a que MockupDD especifica descripciones ejecutables directamente sobre mockups, artefactos que los usuarios finales y clientes pueden comprender.

El enfoque MockupDD, cuyas características técnicas y procedurales se introdujeron con anterioridad, propone un proceso de modelado *liviano* de aplicaciones. A través de los tags atómicos y mediante la aplicación de heurísticas para inferir nuevos elementos del modelo o perfeccionar los ya existentes, MockupDD persigue una estrategia de modelado *just barely good enough* [28]: desde el punto de vista modelador, se pueden aplicar especificaciones directamente sobre los mockups de interfaz de usuario tan pronto como nuevos requerimientos son descubiertos. Las heurísticas ayudan a relacionar y perfeccionar todos estas especificaciones de modelado (por ejemplo, la inferencia de relaciones de datos u transferencia de objetos entre páginas) con el fin de generar modelos lo más completos posible de una manera semi-automática. En resumen, las principales ventajas y motivación de la utilización de MockupDD en comparación con MDWE tradicional son las siguientes:

1. Quita las dependencias estáticas entre el modelado de diferentes concerns. En el contexto del modelado Web (MockupDD Web), se elimina la dependencia lineal que obliga a los modeladores a definir primero un modelo de contenido, luego uno de hipertexto y finalmente otro de presentación.
2. Los requerimientos capturados utilizando mockups se pueden modelar directamente sobre los mismos, evitando la necesidad de utilizar artefactos de requerimientos intermedios - como, por ejemplo, Casos de Uso.
3. Después de capturar y modelar requerimientos sobre la marcha, incluso con presencia de usuarios finales o clientes, una demostración de la aplicación modelada se puede ejecutar instantáneamente para probar la aplicación.

Estas ventajas facilita acortar los ciclos de desarrollo MDWE, cumpliendo así con importantes principios ágiles como *[...] satisfy the customer through early and continuous delivery of valuable software* o *Deliver working software frequently [...] with a preference to the shorter timescale*[10].

3.8. MockRE: Sintaxis alternativas y extensiones de código para los tags

Una de las características fundacionales de MockupDD es el uso de mockups cono base para introducir una técnica de modelado ágil. Al tratarse de un lenguaje intermedio que tanto desarrolladores como stakeholders en general pueden comprender, los mockups permiten reducir el uso de requerimientos textuales, los cuales en general están viciados por *jerga* de negocios. Esta jerga puede presentar a

[10] Principles behind the Agile Manifesto – http://agilemanifesto.org/principles.html

confusiones y errores de interpretación – y, consecuentemente, propiciar errores en el software siendo implementado. Como técnica de modelado sobre mockups y como ya se ha descripto con antelación en detalle, MockupDD propone un *microlenguaje* (extensible) de tags. Habiéndose ya comentado detalladamente la metodología MockupDD aplicado a la Web (MockupDD Web), en esta sección se describirá MockRE [35] (por *Mockup-based Requirements Engineering*), un refinamiento a MockupDD Web el cual mostrará cómo pueden brindarse diferentes sintaxis textuales sobre un mismo conjunto de tags, utilizando un metamodelo común como base. Las ventajas que adiciona MockRE a MockupDD Web tradicional son:

- Mejorar su legibilidad, brindando una sintaxis menos técnica que la ya descripta para los tags.
- Involucrar a los usuarios finales más activamente en le modelado, producto del uso de tags comprensibles por los mismos, los cuales a su vez sean valorables técnicamente.
- Proporcionar un framework para extender rápidamente el comportamiento de los tags con código fuente a fin de refinar la aplicación a mostrar a los usuarios de un modo ágil, utilizando codificación manual.

Todas estas características de MockRE permiten introducir de un modo más intensivo a los usuarios finales en el desarrollo, brindándoles tags que ellos pueden comprender. A su vez, las funcionalidades detalladas que no puedan expresarse con MockupDD - como se verá más adelante en el caso de estudio – y que, por ende, no pueden simularse en el Demo Sandbox, pueden rápidamente codificarse utilizando código client-side para simular el comportamiento de la aplicación en su totalidad.

3.8.1. Motivación

Como se ha visto con antelación, los tags de MockupDD Web se caracterizan por tener diferentes complejidades y poseer, en muchos casos, *microlenguajes* internos – como el caso del tag `Data()` o los tags `Query()` y `Action()` – los cuales pueden volverse relativamente complejos. Por otro lado, en sus formas más simples, varios tags como `Select()`, `Delete()`, `Save()` o el mismo `Data()` pueden ser fácilmente comprensibles por stakeholders promedio, dado que se poseen una complejidad muy reducida. El hecho de que los tags sean comprensibles por clientes o usuarios finales abre las puertas a que ellos validen por sí mismos si los requerimientos especificados por el equipo de desarrollo fueron capturados y expresados correctamente, lo cual evita potenciales errores.

3.8.2. Proceso MockRE

Al ser un refinamiento sobre MockupDD Web, el proceso MockRE es idéntico al ya definido en la Sección 3.7. La única diferencia es que permite integrar a los stakeholders más a menudo en el modelado, gracias a su capacidad para *transformar* tags en un lenguaje legible para los mismos. Para lograr esto, MockRE define por cada tag set un metamodelo interno que representa a los tags de un modo agnóstico a su representación textual concreta. Este metamodelo (graficado en la Fig. 17) se asemeja al modelo de Spec Objects utilizado en el proceso de generación de código de MockupDD Web. A diferencia de MockupDD

Web tradicional, el cual genera los Spec Objects como parte del proceso de generación de código o modelos, el modelo MockRE es generado y construido *al vuelo* por la herramienta de tagging a medida que se van aplicando los tags. Por cada tag set nuevo utilizado en la metodología, una o más clases son *inyectadas* al metamodelo, adicionando funcionalidades. En este nuevo metamodelo, los tags son referidos como `RequirementsFeatures`.

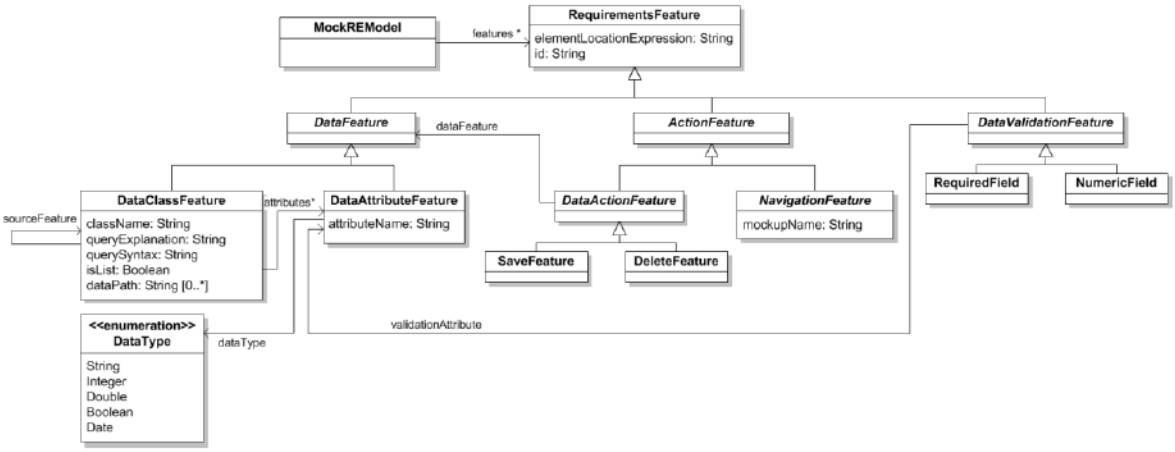


Figura 17. Metamodelo de MockRE.

Tag MockupDD Web	**Tag MockRE orientado a usuarios finales**	**Representación en el metamodelo MockRE**
`Data(<class>)`	`a/an <class>`	`DataClassFeature`, con `className = <class>` and `isList = false`
`Data(*<class>)`	`a list of <class>`	`DataClassFeature`, con `className = <item-type>` y `isList = true`
`Data(<class>.<property>)`	`<class>'s <property>`	`DataAttributeFeature`, con `attributeName = <property>`, relacionado a una `DataClassFeature` con `className = <class>` a través de la asociación `attributes`
`Save(<class>)`	`saves the <class>`	`SaveFeature` asociada a una `DataClassFeature` que especifica el objeto a ser almacenado
`Delete(<class>)`	`deletes the <class>`	`DeleteFeature` asociada a una `DataClassFeature` que especifica el objeto a ser borrado
`Link(<mockup-name>)`	`navigates to <mockup-name>`	`NavigationFeature` con `mockupName = <mockup-name>`

Tabla 1. Tags MockRE, su representación en el metamodelo y su analogía con tags MockupDD Web.

En la Tabla 1 se muestran los dos tipos de sintaxis brindada y su representación interna en el metamodelo. Si bien en MockupDD Web esta representación no era necesaria (dado que hay una sola sintaxis textual para los tags), esta representación es indispensable para MockRE a fin de permitir utilizar ambas notaciones textuales indistintamente.

En MockRE, a diferencia de MockupDD Web en donde los tags son aplicados mayormente por desarrolladores, se brinda una serie de facilidades que permite a usuarios finales introducirlos directamente. Asumiendo que se cuenta con mockups HTML (realizados, por ejemplo, por un equipo de diseño), le herramienta de soporte a MockRE permite importar estos mockups e introducir los tags a través de menues interactivos. Al hacer un click en cada elemento del mockup, se abre un menú interactivo orientado a usuarios finales el cual, dependiendo del tipo de elemento, muestra una serie de tags aplicables. En la Fig. 18 se muestra un ejemplo de estos menues en el idioma inglés, aplicados sobre un mockup que representa la carga de una nueva factura en el contexto de la implementación de una aplicación de facturación clásica. En la versión actual, MockRE ofrece dos grandes tipo de menúes interactivos: *¿Qué es este elemento? – What is this element*, que permite aplicar mayormente tags de contenido (por ejemplo `a/an <class>`) o *¿Qué es lo que hace este elemento? – What does this element do?*, el cual permite aplicar tags de navegación, interacción y manipulación de contenido.

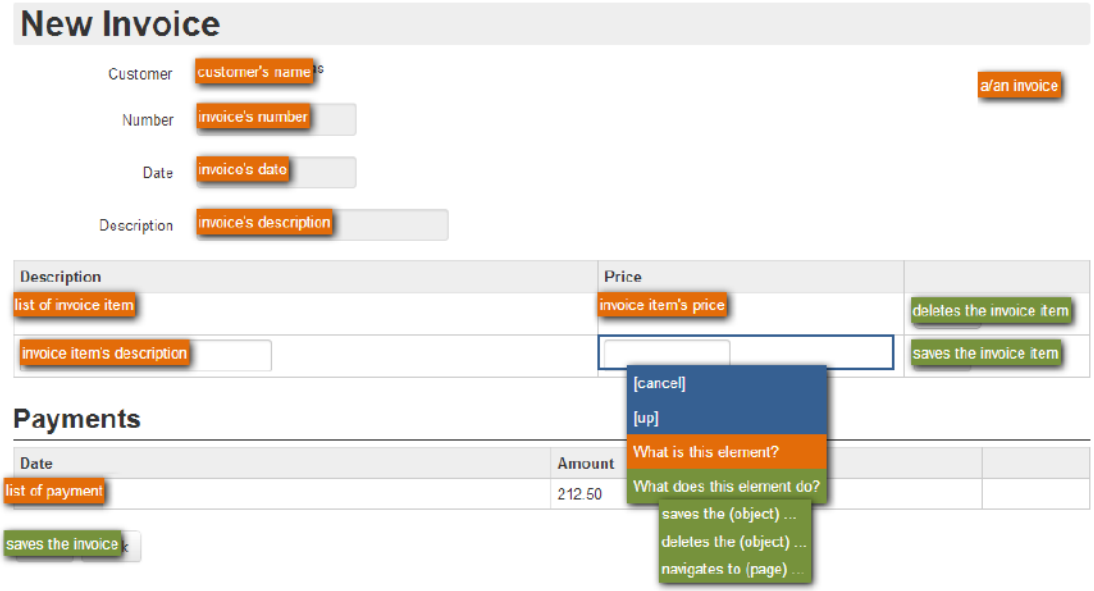


Figura 18. Aplicación interactiva de tags orientada a usuarios finales.

Bajo este contexto planteado, un usuario final puede comenzar aplicando los tags por sí mismo y luego el equipo de desarrollo puede refinarlos tanto como sea necesario, utilizando la herramienta en *modo de desarrollo*. Este modo representa los tags utilizando una sintaxis textual formal, la cual expone todas las características internas descriptas en el metamodelo. Esto le permite a los desarrolladores refinar al máximo posible los tags y su comportamiento. En lugar de utilizar la sintaxis de MockupDD Web, se utiliza una sintaxis completamente técnica y sin ambigüedades a fin de que los desarrolladores puedan manipular internamente los tags como lo deseen. Esto diferencia MockRE de MockupDD Web, el cual utiliza una sintaxis intermedia entre lo técnico y lo comprensible por usuarios finales – y por lo tanto, con ciertas ambigüedades que deben ser refinadas, como ya se mostró con antelación. En la Fig. 19 se muestran los mismos tags denotados en la Fig. 18 pero en *modo de desarrollador*.

La sintaxis de los tags en modo de desarrollo está comprendida simplemente por el tag set y nombre del tag aplicado (siguiendo la forma `<tag-set>::<tag-name>`) y luego contenido en formato JSON[11] que describe valores para cada una de las propiedades del metamodelo.

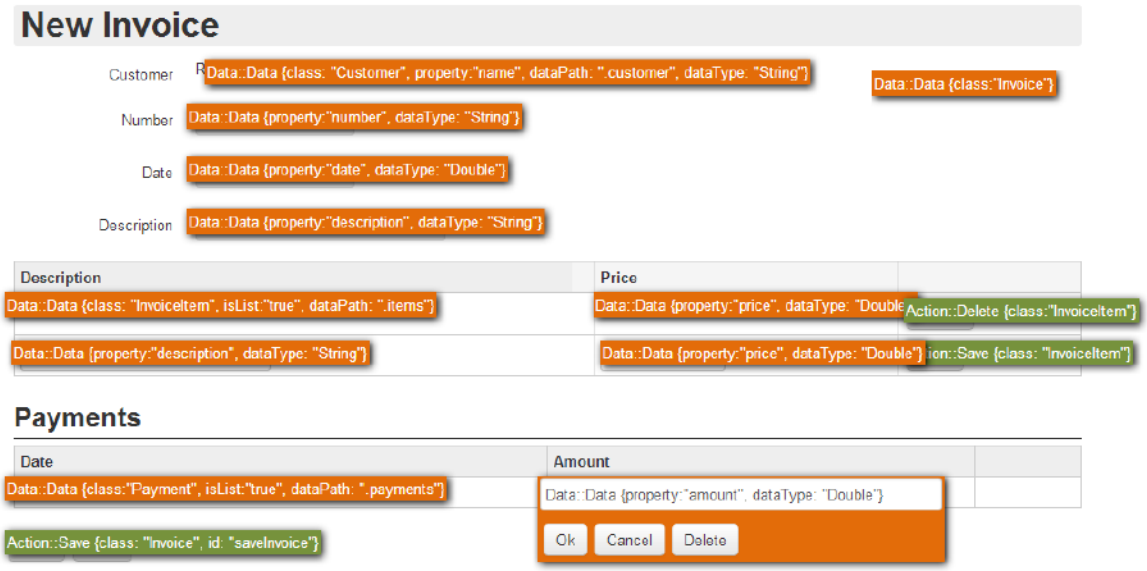


Figura 19. Tags de la Fig. 18 en modo de desarrollo.

Al igual que en MockupDD Web, una vez refinados los tags puede correrse una versión de demostración de la aplicación al instante haciendo uso de una las herramientas de soporte. En adición, también puede utilizarse codificación directa para *inyectar* funcionalidad adicional a estos tags. Más puntualmente, cada tag ofrece ciertos puntos de refinamiento en los cuales los desarrolladores pueden escribir código personalizado para realizar cualquier acción, siempre desde el punto de vista client-side – es decir, desde el browser. Esto permite extender rápidamente la aplicación simulando los requerimientos que no pueden ser expresados con tags a fin de que una versión funcional completa de la aplicación, al menos para usuarios finales, pueda ser probada rápidamente. Detalles de implementación de MockRE serán descriptos más adelante, junto con aquellos pertenecientes a MockupDD Web.

3.8.3. Conclusión

La metodología MockRE busca extender MockupDD Web para facilitar la trazabilidad de requerimientos y participación de los usuarios finales en el modelado. Éstos pueden participar desde el principio del desarrollo introduciendo tags sin asistencia de desarrolladores, con un lenguaje y menúes interactivos que pueden comprender fácilmente. Asimismo, pueden probar una versión corriente de la aplicación con lo modelado por ellos mismos hasta el momento. Dado que los tags más importantes son introducidos o, en el

[11] JavaScript Object Notation – http://json.org/

peor de los casos, enteramente comprensibles por usuarios finales, se asegura que los mismos sean totalmente validables

Una vez alcanzado el límite de lo modelable por usuarios finales, los desarrolladores pueden utilizar la misma herramienta en modo desarrollo para refinar los tags tanto como sea necesario y posible. En este punto, también puede correrse una versión prototípica de prueba de la aplicación para que sea validada y verificada por usuarios finales. Finalmente, cualquier refinamiento extra que sea necesario introducir puede ser adicionado utilizando codificación final, sin romper la abstracción ya introducida por los tags (la cual también representa requerimientos trazables). Esto representa varias mejoras en lo relativo a extensibilidad y agilidad para implementar aplicaciones funcionales validables por usuarios finales en comparación con MockupDD Web tradicional.

4. Herramientas e Implementación de MockupDD Web

En esta sección se presentarán las herramientas necesarias para soportar las etapas del proceso MockupDD como se ha descrito anteriormente – haciendo principal foco en MockupDD Web, la especialización de MockupDD descripta hasta el momento. Primero se describirá las funcionalidades y el rol de cada herramienta en la metodología. Luego, se comentarán aspectos técnicas y de implementación detallados para cada una.

4.1. Herramientas de soporte

Casi todos los pasos de la metodología MockupDD Web (representados en la Fig. 4) son asistidos con herramientas específicas. A continuación se enumeran las diferentes herramientas de apoyo a la metodología:

- **Herramienta Mockup-to-HTML (MHT)**: introducida previamente en [6], esta herramienta permite procesar archivos de mockups construidos a partir de herramientas tradicionales de mockuping como Balsamiq o Pencil y generar código HTML subyacente para comenzar con el modelado. Al mismo tiempo, la herramienta MHT genera el modelo SUI y asocia cada componente a su elemento HTML correspondiente (Fig. 4, Paso 2). De este modo, se evita escribir el código HTML desde cero a partir de los mockups, lo cual agiliza las primeras etapas del proceso de modelado.
- **Herramienta Interactiva de Tagging (Interactive Tagging Tool, ITT)**: Permite la importación de mockups HTML (que pueden haber sido obtenidos con la herramienta MHT o bien escritos por desarrolladores o diseñadores). Una vez importados, la ITT facilita la selección de los elementos relevantes de cada de cada mockup (transformándolos en conceptos SUI) y la aplicación de tags sobre los mismos, asistiendo en la etapa de Detección de Widgets Relevantes (Fig. 4, Paso 2). De este modo, esta herramienta funciona como una suerte de *frontend de modelado* para la metodología, creando los modelos subyacentes y permitiendo el modelado interactivo directamente sobre mockups. Para ello, la ITT puede funcionar en dos modos: *Descubrimiento de Widgets* (para construir el SUI interactivamente descubriendo widgets relevantes mockups HTML) y *Tagging de Widgets* (para aplicar tags sobre widgets ya descubiertos). Un *screenshot* de la herramienta puede observarse en la Fig. 20.
- **Demo Sandbox (DS)**: Una vez que los enriquecimientos aplicados a través de tags son suficientes para satisfacer parcial o totalmente los requerimientos del Sprint actual, la herramienta de Demo Sandbox permite generar rápidamente una versión corriente de la aplicación la cual puede ser probada por usuarios finales. La herramienta DS no es dependiente de la metodología MDWE subyacente y no requiere ningún tipo de deployment de la aplicación, sino que *emula* su funcionamiento, utilizando un *sandbox* el cual enriquece los mockups HTML estáticos para simular el comportamiento expresado por los tags.

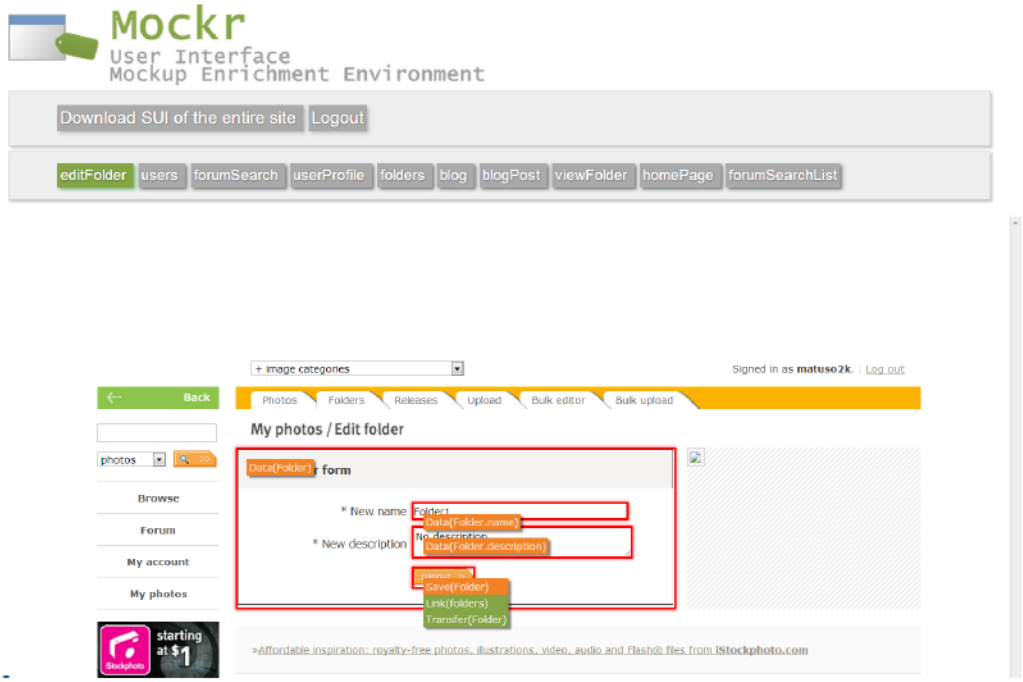


Figura 20. Captura de pantalla de la Interactive Tagging Tool.

La ITT y la herramienta DS realizan las tareas más importantes en el proceso de MockupDD Web. La Herramienta Interactiva de Tagging también proporciona la desambiguación de tags y propone refinamientos como se comentó antes, así como también puede ejecutar la demostración de la aplicación corriente invocando a la herramienta DS y generar modelos MDWE para metodologías conocidas como WebML.

4.2. Aspectos de Implementación

En esta subsección se describirán con mayor grado de detalle aspectos de implementación relativos a cada una de las herramientas de soporte de MockupDD Web comentados en la sección anterior.

4.2.1. Modelos SUI

Para permitir su completa trazabilidad respecto a los principales artefactos de requerimientos (mockups, los modelos SUI deben construirse y ser *mapeados sobre* un mockup en su representación. Esto quiere decir que cada uno de sus componentes debe estar relacionado con un elemento particular del mockup que le dio origen. Desde un punto de vista técnico y en el contexto de MockupDD Web, si se utilizan mockups HTML como se comenta antes, esta correspondencia se conserva con una expresión XPath[12] que define el elemento HTML de origen la cual se incluye en el modelo SUI a través de la propiedad `mockupRepresentation` de `Widget`. Si por otro lado se utiliza una herramienta de mockuping soportada (como Balsamiq o Pencil), se calcula un identificador a partir del código fuente de el mockup y se asocia al elemento SUI; Este *id* se

[12] XML Path Language – http://www.w3.org/TR/xpath

genera de una manera que pueda identificar de forma inequívoca el elemento asignado en la representación del mockup – por ejemplo, en el caso Balsamiq se obtiene concatenando el nombre de el mockup y el nombre del elemento.

4.2.2. Herramienta Interactiva de Tagging

La ITT consiste en una Aplicación Web que enriquece mockups HTML estáticos con el comportamiento requerido para resaltar y marcar elementos DOM interactivamente. Esto permitirá al usuario de esta aplicación denotar qué elementos de la UI son importantes desde el punto de vista de los requerimientos y, de este modo, construir el SUI interactiva e iterativamente. Dependiendo del tipo de elemento DOM que se marque, se generará un tipo de `Widget` SUI particular. Por ejemplo, el marcado de un tag `<a>` provocará la creación de un `Widget Link`, mientras que al marcarse un elemento DOM de tipo `<input type="text">` se creará un `Inputbox`. Para poder implementar este marcado interactivo, la herramienta proporciona una API en el lado del servidor la cual es invocada bajo demanda por el client-side a medida que se van marcando elementos. Para cada elemento se calcula su expresión de selección XPath y se envía al server-side para que sea agregado al SUI para la página (`Screen` SUI) actual.

Cuando se trabaja con mockups construidos utilizando herramientas mockup compatibles (como Balsamiq o Pencil), la herramienta MHT permite generar automáticamente una instancia del SUI y su representación HTML para cada mockup utilizando un componente llamado *Motor de Procesamiento de Mockups* (Mockup Processing Engine o MPE) [36]. El MPE está compuesto por un conjunto de elementos de procesamiento individuales encadenados – un esquema gráfico del flujo atravesado al procesar mockups puede observarse en la Fig. 21. Los componentes de procesamiento más importantes que son atravesados, tomando como punto de partida uno o más archivos de mockups digitales construidos con una herramienta, son los siguientes:

1. *Widget Parser:* Este procesador toma archivos de mockup como entrada e identifica todos los widgets individuales que están presentes en el mismo. Su salida es una lista sin ordenar pero uniforme de widgets, agrupados por mockup. Dado que el formato fuente mockup es específico para cada herramienta, existe un Widget Parser para cada herramienta soportada.
2. *Containment Detector:* Este procesador toma la lista de widgets sin clasificar del *Widget Parser* y descubre las relaciones de contención entre ellos. Como la mayoría de las herramientas digitales de *mockuping* persisten mockups como una simple lista de los widgets con coordenadas específicas junto con su ancho y alto, este procesador permite descubrir la estructura jerárquica de los elementos de cada mockup.
3. *Layout Inferer:* Este procesador toma la lista de widgets jerarquizada del *Containment Detector* y analiza la distribución gráfica interna de cada `CompositeWidget` para inferir un tipo de layout específico y la distribución de sus widgets de acuerdo a al mismo. La inferencia se lleva a cabo mediante la aplicación de heurísticas sobre la disposición gráfica de los widgets en dos pasos: (1) un pre-análisis de la distribución de widgets, infiriendo el layout que mejor cuadre pare los mismos y (2) la determinación la disposición de cada widget de acuerdo con el layout elegido en el

paso anterior. Si bien este proceso no es relevante desde el enfoque de modelado en sí, es obligatorio para obtener una estructura visual HTML adecuada. Detalles adicionales sobre el modelado de diseño y la inferencia se pueden encontrar en [36].

Con respecto a la representación del metamodelo SUI (ver Fig. 6), se consideraron varias opciones diferentes. Como se muestra en la figura, se decidió por modelar los tags y sus aplicaciones como clases separadas y asociadas al SUI model. Otra alternativa que se consideró fue el uso de estereotipos UML sobre los `Widgets` para representar a los tags – es decir, que los tags sean definidos y expresados a partir de un Profile UML. Sin embargo, la representación en base a un Modelo de Clases que se eligió permite, por un lado, reproducir exactamente cómo los tags han sido modelados internamente en las herramientas de soporte de MockupDD anteriormente mencionadas. Al mismo tiempo, esta representación permite tener el mismo nivel expresivo que utilizando estereotipos UML, omitiendo aquellas características de estos últimos que no son implementables en comunes orientados a objetos como Java o C#. Finalmente, la estructura del metamodelo se asemeja a las diferentes representaciones textuales del modelo SUI, como el XSD Schema que define su estructura en XML[13].

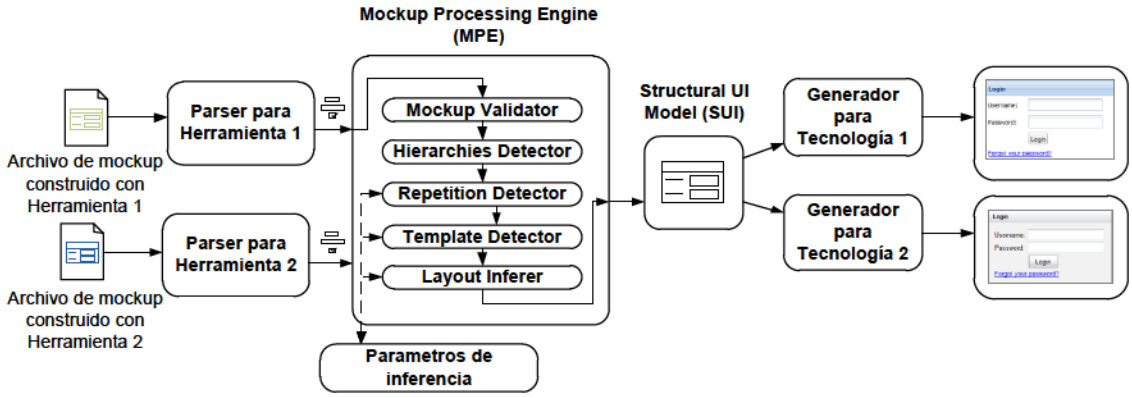


Figura 21. Flujo de procesamiento de mockups en el contexto del MPE.

Con respecto al procesamiento y refinamiento de tags, un *Tag Inferer* se utiliza para analizar los tags ambiguos y proponer posibles soluciones a los modeladores. Si más de una solución de refinamiento es posible, la ITT muestra un menú emergente para seleccionar la opción deseada por el modelador. Por otro lado, si sólo se detecta una mejora posible, ésta se realiza de forma automática.

4.2.3. Demo Sandbox

Como se comentó anteriormente, la herramienta DS permite generar rápidamente una versión corriente de la aplicación a fin de que sea probada por usuarios finales. Para lograr esto, la herramienta presenta los mockups HTML estáticos originales y realiza un conjunto de uno o más enriquecimientos del lado del cliente por cada tag (de acuerdo a la semántica de cada uno) para permitir a desarrolladores o usuarios finales interactuar con un prototipo de lo que será la aplicación final. El código generado por DS no es

[13] MockupDD XSD Schema – http://agilemdd.lifia.info.unlp.edu.ar/mockupdd/sui.xsd

necesariamente eficiente ni escalable (por ejemplo, parte de su implementación se basa en la manipulación del modelo DOM para actualizar la interfaz de usuario y LocalStorage HTML para simular persistencia), pero puede ser generado muy rápidamente (sin demoras ni costo de deployment) para mostrar una simulación de lo que será el comportamiento final de la aplicación.

Para poder generar una versión de demostración de la aplicación corriente, la herramienta DS invoca la generación de Spec Objects para los tags actuales. Luego, ejecuta cierto fragmento de código client-side para cada Spec Object particular; cada uno de estos fragmentos agrega al modelo DOM subyacente la semántica necesaria para emular la aplicación final en el browser. Para esto, cada Spec Object debe tener lo que se conoce como un *client-side spec*, un fragmento de código JavaScript el cual describirá su semántica de acuerdo a sus variables internas y relación con otros tags.

La ejecución en el DS es llevada a cabo recursivamente, siguiendo un recorrido BFS sobre el árbol de Spec Objects (SOs). Cada uno de estos SOs define el orden y tipo de evaluación de sus subordinados o relacionados dependiendo de sus requerimientos. A su vez, dado que cada Spec Object conoce a su elemento SUI subyacente (y, por lo tanto, al elemento DOM sobre el cual fue aplicado, ver Fig. 13), los enriquecimientos son evaluados directamente sobre estos elementos – por ejemplo, adosando eventos, manipulando su contenido y estructura, etc. A continuación, se describirá la implementación concreta de los enriquecimientos sobre el DOM para los Specs Objects introducidos en las Figs. 13 y 14:

- **`ReadSpec`**: Este SO se comporta de modo diferente dependiendo de su configuración interna. El primer paso en la ejecución client-side de un `ReadSpec` es la obtención de las instancias de datos a mostrar. Eso se realiza de diferentes formas dependiendo de su configuración interna. En primer lugar, si su propiedad `list` es `true`, entonces obtiene a partir de la capa de persistencia prototípica (en el caso de la implementación actual, el LocalStorage de HTML), la lista completa de elementos de clase `className`. Por otro lado, si esta misma propiedad tiene el valor `false`, el `ReadSpec` obtendrá una sola instancia de datos desde su origen de datos específico, comunicándose con su `DataRelatedSpec` relacionado a través de la asociación `dataSource` – dado que no puede obtenerla por sí mismo. También puede ocurrir el caso en que `list` sea `true` y se tenga una `DataRelatedSpec` relacionada, con lo cual, a pesar de referenciar una lista de elementos, el `ReadSpec` obtendrá su lista de instancias a través de su `DataRelatedSpec` asociada. Finalmente, de haberse provisto la propiedad `navigationPath`, las instancias a mostrar serán aquellas que estén unidas a través del camino de una o más asociaciones de datos especificados por la propiedad. Una vez obtenidos el o las instancias de datos que serán mostradas en la interfaz de usuario, un `ReadSpec` itera por cada uno de sus `ReadPropertySpec` (a través de la asociación `properties`), cuya acción por defecto es descripta debajo. Las instancias de datos obtenidas directa o indirectamente por las `ReadSpec` son almacenadas internamente y pueden ser utilizadas por otros Spec Objects.
- **`ReadPropertySpec:`** Este Spec Object recibe un objeto de datos proveniente de su `ReadSpec` asociado. Su ejecución consiste en leer una propiedad dada de ese objeto y hacer

visible su valor en la interfaz de usuario, manipulando el DOM. En general, esta actividad consiste en rellenar el DOM con un nodo de texto con el valor de la propiedad, aunque puede consistir en asignar un atributo concreto de un nodo DOM existente (por ejemplo, un `<input>`). Cada caso puntual se trata de un modo diferente y es abstraído por este SO.

- **`SaveSpec`**: La ejecución de este Spec Object consiste en instanciar un objeto de determinada clase y almacenarlo utilizando la capa de persistencia prototípica. En primer lugar, un `SaveSpec` crea un objeto JavaScript vacío (es decir, sin ninguna propiedad asignada). Luego, ejecuta en orden cada uno de sus `SavePropertySpecs` internos (los cuales conoce a través de la relación `properties`). Finalizada esta ejecución, el objeto en sus inicios sin ninguna propiedad se asume *hidratado* por cada una de las propiedades que le asignó cada `SavePropertySpec` (descripto más adelante). En este punto, se invoca la capa de persistencia para que almacene el objeto y se dispara un evento de actualización automática. Este evento provocará la actualización de todas las listas y elementos en la UI que puedan llegar a estar involucrados con el nuevo objeto existente en la aplicación.
- **`SavePropertySpec`**: Este SO recibe un objeto JavaScript vacío o parcialmente completo por la ejecución de `SavePropertySpecs` anteriores y asigna un valor a una nueva propiedad del objeto. Para lograr esto, obtiene el valor desde el elemento DOM que tiene asociado (por ejemplo, un `<input>`) y lo asigna al objeto con el nombre de propiedad `propertyName`.
- **`AssociateSpec / DissociateSpec:`** Estos tipos de SOs toman objetos de `DataRelatedSpecs` y los asocian o disocian en el contexto de cierta relación particular. Su ejecución consiste en solicitar los dos objetos a relacionar desde los `DataRelatedSpecs` referenciados a través de las asociaciones `sourceSpec` y `destinationSpec` e invocar la capa de persistencia prototípica indicando que los mismos se asociarán o disociarán (en este último caso, de haber sido previamente asociados) utilizando la relación `associationName`.
- **`ActionSpec:`** La función de este tipo de Spec Object consiste simplemente en agrupar diferentes tipos de acciones ejecutables (subclases de `AtomicActionSpec`). Su ejecución consiste en enriquecer su elemento DOM subyacente para que, al ocurrir un evento dado (usualmente un click o *tap* sobre el mismo), se ejecuten sincrónica y linealmente cada una de las acciones que referencian a través de la asociación `actions`.

Al cargarse el mockup HTML en la herramienta DS, la ejecución es llevada a cabo a través del siguiente workflow

1. El primer paso consistirá en obtener toda la información requerida por los `DataRelatedSpec` con `list = true` y `dataSource` nulo, dado que sus fuentes de datos son autónomas y se obtienen invocando la mecanismo de persistencia prototípico.
2. El segundo paso consistirá en proveer a todos los `ReadSpecs` que dependan de los primeros para obtener sus instancias de datos. Estas dependencias de datos (definidas por

las relaciones entre los diferentes `ReadSpecs` a través de la asociación `dataSource`) se resolverán es cascada hasta que todos los `ReadSpecs` tengan los datos requeridos y puedan plasmarlos en la interfaz de usuario.

3. Una vez concluidos estos dos pasos, se ejecutarán los `ActionSpecs` que asocien los eventos necesarios en la interfaz de usuario a fin de ejecutar las acciones necesarias cuando se requiera.

Atravesados estos tres pasos principales, la interfaz de usuario queda totalmente enriquecida, proveyendo y simulando todas las funcionalidades que la aplicación final implementará.

4.2.4. Demo Sandbox extendido para MockRE

Como se ha comentado con antelación, MockRE permite extender el comportamiento de los tags aplicando una estrategia similar al patrón de diseño Strategy [37]. Desde el punto de vista técnico, MockRE es implementado como una versión mejorada del Demo Sandbox ya introducido para MockupDD Web. A diferencia de la primera versión, esta fue planteada de un modo más extensible. Al igual que en su versión original, la aplicación es simulada en el client-side utilizando JavaScript y diferentes tecnologías en el browser. Sin embargo, en este caso, la implementación de cada tag es llevada a cabo por una clase o prototipo específico, codificado en JavaScript. Todos los tags son subclases de una clase `Tag` general y ofrecen una técnica no intrusiva para redefinir parte del comportamiento client-side del tag, sin necesidad de romper su abstracción por completo.

Siguiendo este paradigma de enriquecimiento no intrusivo (como el de la popular librería JavaScript jQuery[14]), el Demo Sandbox de MockRE permite obtener los objetos que representan a los tags y enriquecer ciertos aspectos de los mismos utilizando un sistema de registración de acciones. Por ejemplo, en el ejemplo de la Fig. 19, en la cual se describe un mockup correspondiente a una pantalla de carga de facturas en el contexto de una aplicación de facturación, podría implementarse una regla de validación la cual fuerce a que una factura cargada tenga al menos una línea. Esto podría lograrse con el código descripto en la Fig. 22, en la cual se extiende el tag Save que permite persistir la factura que se está cargando, introduciendo la regla mencionada. El método `mockreEngine.getFeature()` permite obtener el tag en cuestión (identificado como `saveInvoice` por los desarrolladores). Luego invocando el método `.on()` con el parámetro `before` se registra una acción a llevarse a cabo antes de efectuar la persistencia. En el caso de ejemplo, se eligió mostrar un mensaje de alerta utilizando el método `alert()` del browser, pero podría haberse utilizando cualquier otro método de cualquier otra librería client-side incluida en el proyecto.

[14] jQuery – http://jquery.com

```
mockreEngine.getFeature('saveInvoice').on('before', function(invoice) {
	if (invoice.lines.length == 0) then {
		alert('The invoice must have at least one line');
		return false;
	}
	return true;
});
```

Figura 22. Código que extiende un tag Save existente para introducir una validación.

Este esquema extensible permite aprovechar la abstracción de los tags y, a su vez, extender la solución con código manual tanto como se requiera. Por ejemplo, podría extenderse el tag `saveInvoice` anteriormente comentado para que persista objetos a través de una API RESTful externa o cualquier otra acción deseada.

5. Caso de estudio

Con el objeto de mostrar la metodología MockupDD Web en acción, en esta sección se describirá un caso de estudio completo en el cual se aplicó la misma para el desarrollo de una aplicación data-intensive concreta. La aplicación en cuestión fue el ya brevemente introducido site Photo Stock, el cual en esta ocasión será descripto en mayor grado de detalle, contemplando gran parte de sus User Stories y mockups. El desarrollo será dividido en 3 Sprints, siguiendo el Proceso MockupDD adaptado desde la metodología Scrum. Cada iteración será tratada en una subsección diferente. Por una cuestión de simplicidad y claridad durante el transcurso de la descripción del caso, no todos los elementos de los mockups serán asociados y taggeados.

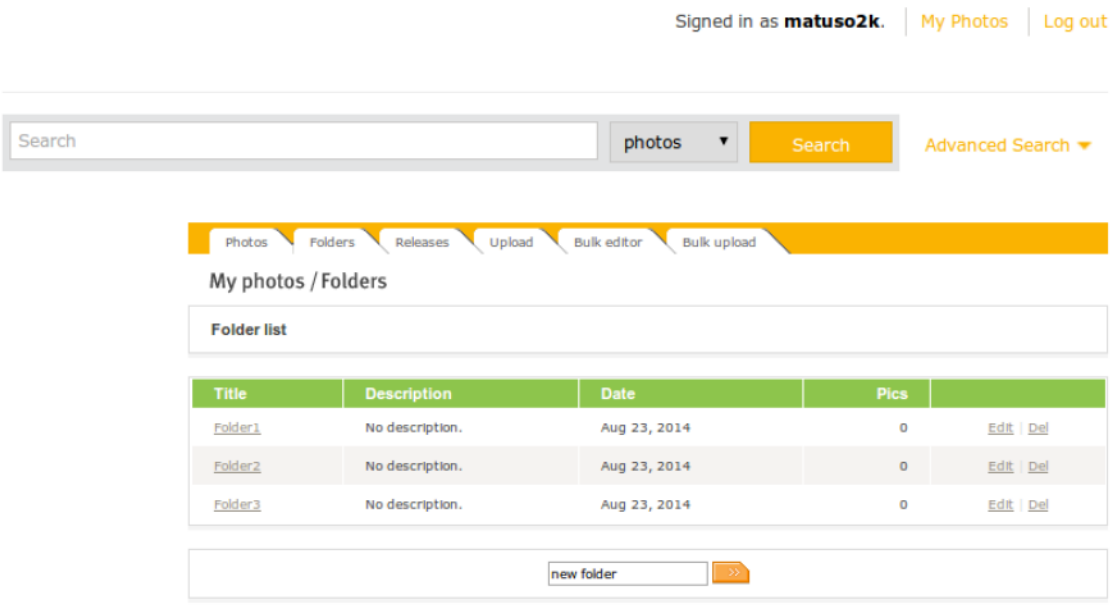
5.1. Sprint 1

Siguiendo el proceso de MockupDD (graficado en la Fig. 16), se comenzará el Sprint con la definición de User Stories y mockups asociados. Se utilizarán mockups HTML dado que permiten trabajar sobre una base segura de requerimientos de presentación y, a su vez, son enteramente reusables por la metodología, la cual permite *dinamizarlos* utilizando la estrategia de tagging y las herramientas ya detalladas.

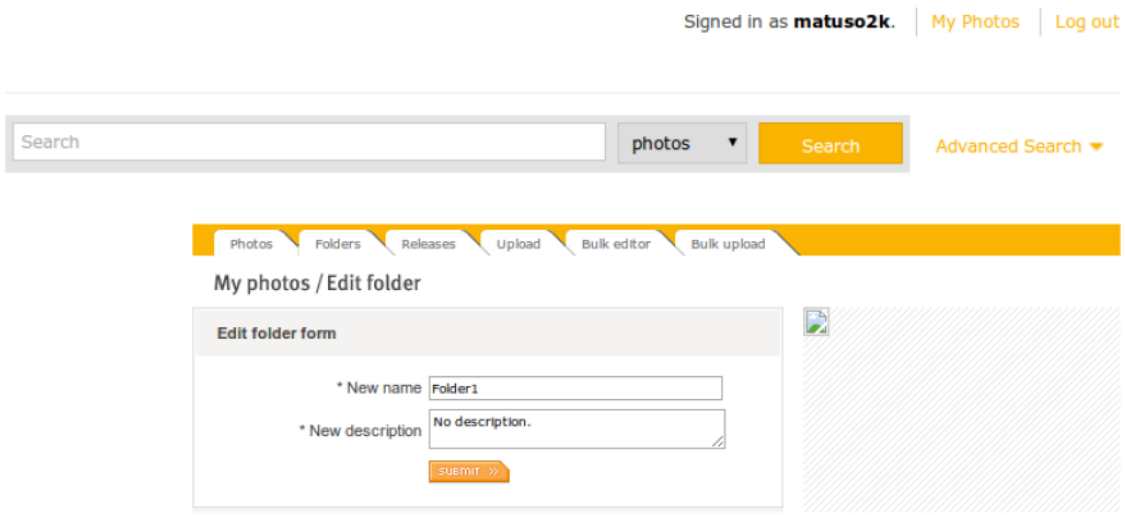
En la primera iteración se acuerda con los stakeholders la implementación de la funcionalidad de organización y carga de fotos a la aplicación, la cual será una funcionalidad indispensable para la monetización de la misma. Luego del Sprint Meeting inicial con el equipo de desarrollo, surgen las siguientes US las cuales conformarán el Sprint Backlog de esta iteración:

- **US1.** Como Usuario, quiero poder listar mis carpetas, crear nuevas carpetas o editar carpetas existentes, en las cuales se almacenaran fotografías.
- **US2.** Como Usuario, debo poder eliminar carpetas existentes con o sin fotografías almacenadas.
- **US3.** Como usuario, debo poder subir nuevas fotografías a carpetas previamente creadas.
- **US4.** Como usuario, debo poder ver las fotografías subidas dentro de una carpeta, editarlas y borrarlas.

Siguiendo el proceso, el siguiente paso será la definición de mockups para cada una de las US descriptas, para luego pasar a la etapa de importación y anotación de los mismos. Los mockups confeccionados para cada US se describen en las Figs. 23.a y 23.b.

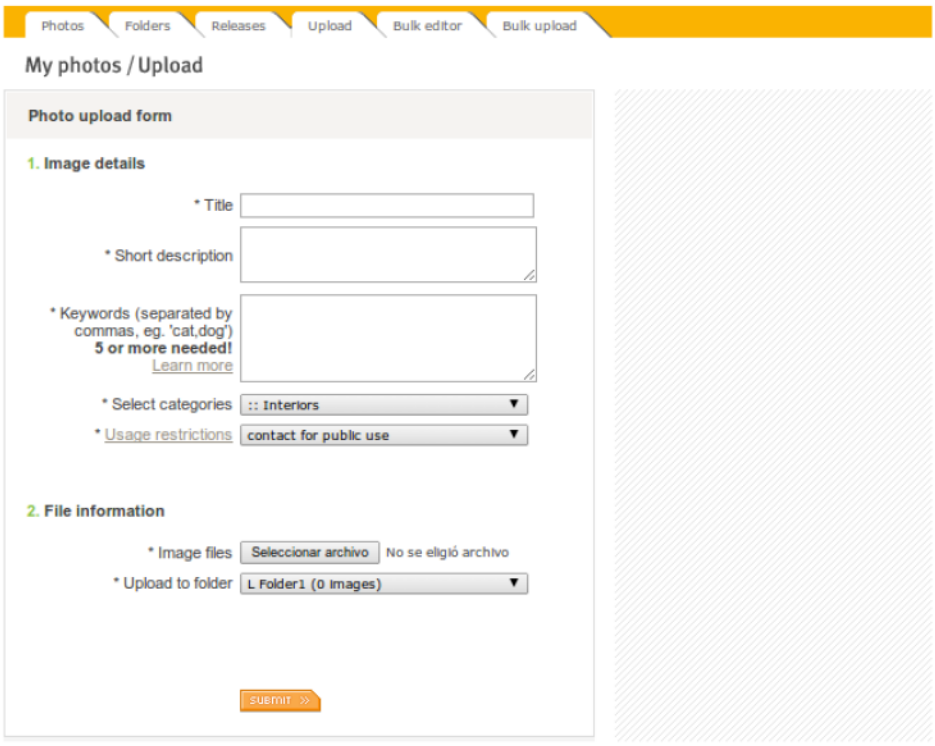


Mockup 1 – Permite el listado de carpetas y su eliminación (implementa US1 y US2)

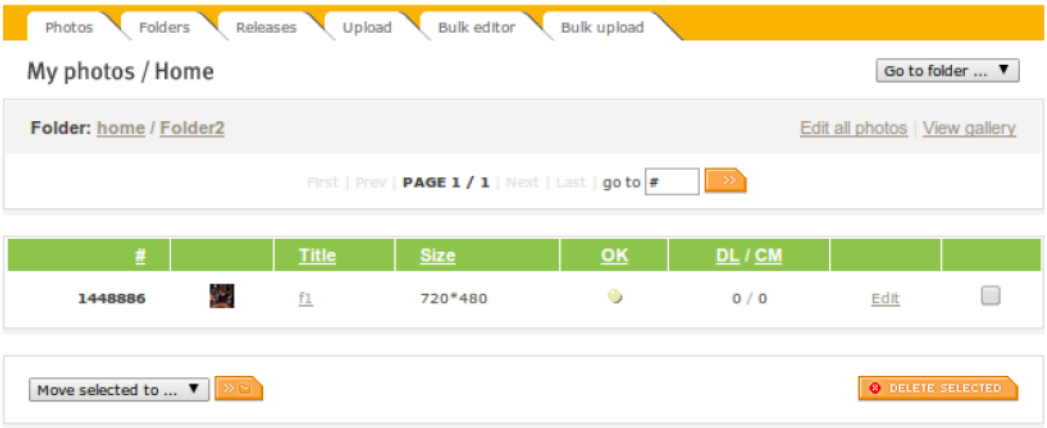


Mockup 2 – Permite la creación de nuevas carpetas y la edición de carpetas existentes (implementa US1)

Figura 23. Mockups necesarios para la implementación del Sprint 1.



Mockup 3 – Permite subir nuevas fotografías a una carpeta ya creada (implementa US3 y US4)



Mockup 4 – Permite visualizar y eliminar fotografías en una carpeta dada (implementa US4)

Figura 23. Mockups necesarios para la implementación del Sprint 1 (cont.).

Una vez definidos los mockups y su asociación con las US correspondientes, se procede a su tagging. Las versiones anotadas de ambos mockups pueden observarse en la Fig. 24. No se entrará en detalles acerca de la descripción semántica de los tags en los Mockups 1 y 2 dado que los mismos ya fueron descriptos en el ejemplo de la Sección 3.3.

Respecto al Mockup 3 (Fig. 23), el cual consiste en una pantalla de upload de una fotografía particular, pueden describirse varios detalles relativos a sus tags (ver Fig. 24). En particular, varios de ellos describen no sólo los atributos de la entidad de negocios que representa a una fotografía (`Photo`), sino varias clases

relacionadas. El tag `Data(Photo.category -> Category.name)` aplicado sobre un widget de tipo `DropdownList` indica que esa lista mostrará las instancias de `Category` en la aplicación (una nueva entidad de negocios que define la categoría de una fotografía, descripta por primera vez en este tag). A su vez, este tag también describe la relación de una `Photo` con su `Category` a partir de su asociación denominada `category`. Lo mismo ocurre con el tag `Data(Photo.usageRestriction -> UsageRestriction.name)`, el cual define `UsageRestriction`, entidad la cual representa las condiciones de uso de una fotografía. El tag `Data(Photo.folder -> Folder.name)` sigue el mismo patrón, relacionando una fotografía a su carpeta (`Folder`), entidad la cual fue definida en los mockups anteriores. Todas estas propiedades son asumidas como de tipo `String` (dado que no fue explicitado un tipo de datos concreto) excepto por el caso de la imagen de la fotografía (especificada en el tag `Data(Photo.image:Blob)`, el cual es un tipo de datos binario – representado en la fachada Web como un archivo.

Por último, en el Mockup 1, los tags `Link(Photos)` y `Transfer(Folder)` provocan una navegación al Mockup 4 (llamado `Photos`), el cual muestra la lista de fotografías en una carpeta concreta. En este mockup se hace explícito la obtención de la lista de fotografías desde la carpeta pasada por parámetro a través del tag `Data(Folder.photos -> *Photo)`. En el mockup se muestran algunas propiedades de la fotografía (entre ellas, la propiedad `id` a través del tag `Data(Photo.id)`, la cual es una propiedad generada automáticamente por la herramienta). El botón *Edit*, instanciado por cada fotografía, navega hacia la pantalla denominada `Photo` (Mockup 3), la cual recibe la fotografía como parámetro y permite editarla – para denotar esta acción se utilizan los tags `Link(Photo)` y `Transfer(Photo)`. El `Checkbox` de la derecha, marcado con el tag `Select()`, determina que el mismo se utilizará para seleccionar las instancias de `Photo` que serán producto de alguna acción en la UI a posteriori. Esta semántica es utilizada por tag `Delete(photoList:Folder)` el cual expresa que, al hacer click sobre el botón *Delete Selected,* se disparará el borrado de todas las instancias de `Folder` elegidas utilizando los checkboxes en la `Repetition` de fotografías (cuyo id de widget en el modelo SUI es `photoList`). Finalmente, el tag `Associate(photoList:Photo, folderList:Folder, folder)` determina que las carpetas seleccionadas en la `Repetition photoList` serán asociadas con el `Folder` elegido actualmente en la `DropdownList folderList`, y que la asociación se hará a través de la relación `folder` desde `Photo` a `Folder`.

Con los tags agregados en la Fig. 24, se implementa por completo la lógica necesaria para satisfacer las tres US correspondientes al Sprint 1. Los usuarios finales pueden probar el prototipo totalmente funcional de la aplicación al instante utilizando el Demo Sandbox.

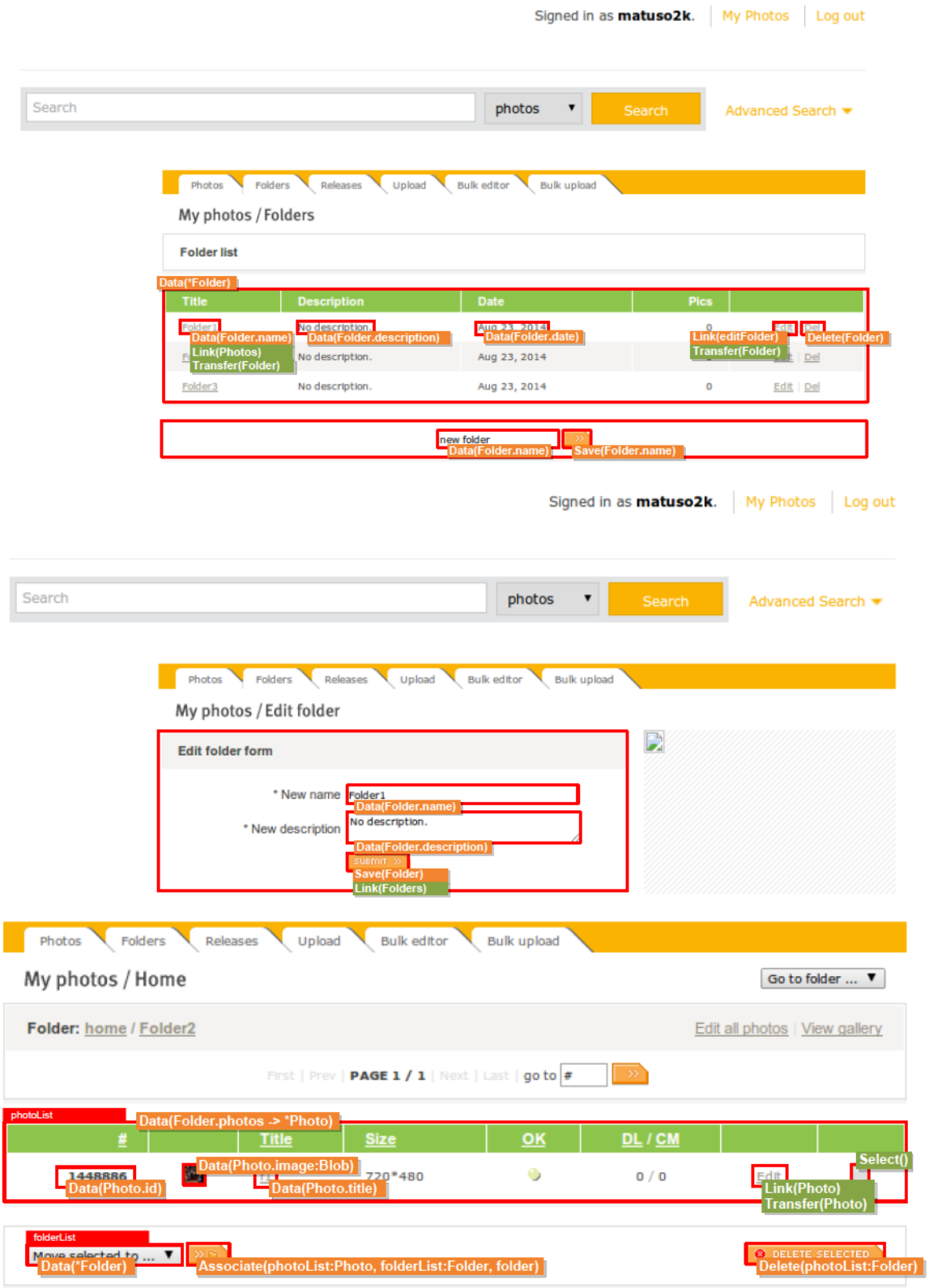


Figura 24. Mockups de la Fig. 19 taggeados, implementando por completo cada uno de los issues del Sprint 1.

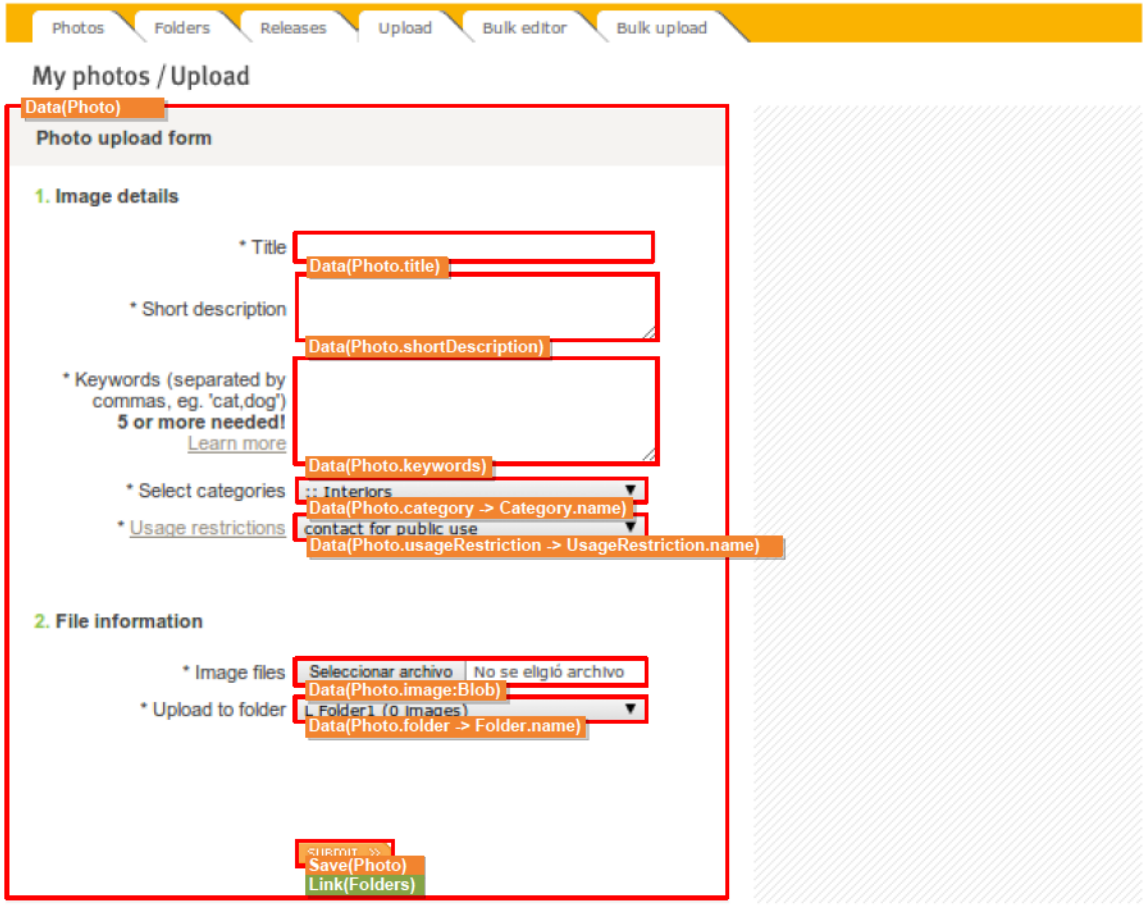


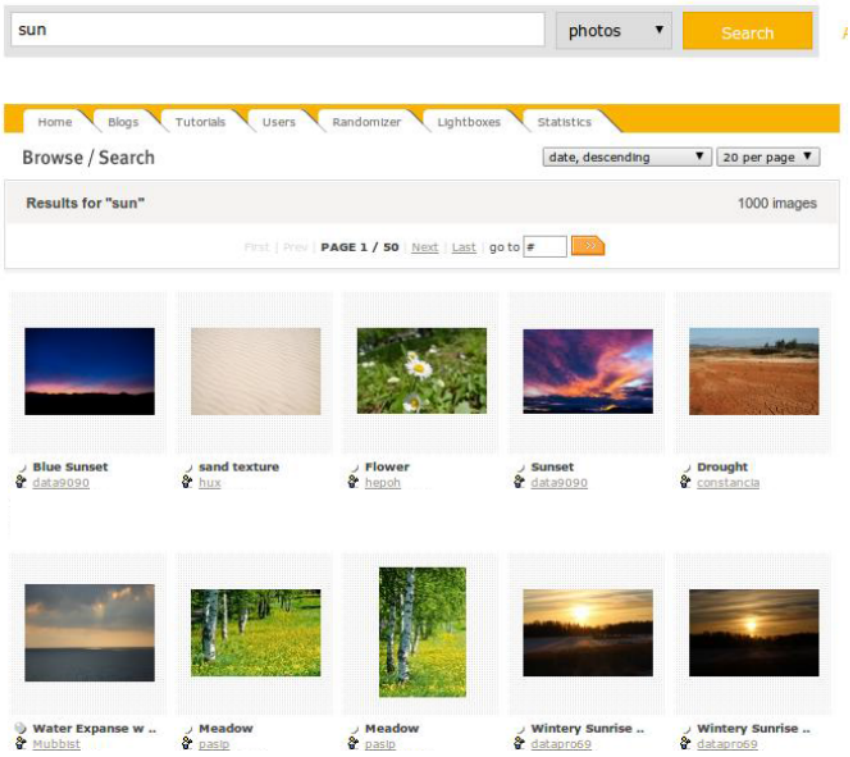
Figura 24. Mockups de la Fig. 19 taggeados, implementando por completo cada uno de los issues del Sprint 1 (cont.).

5.2. Sprint 2

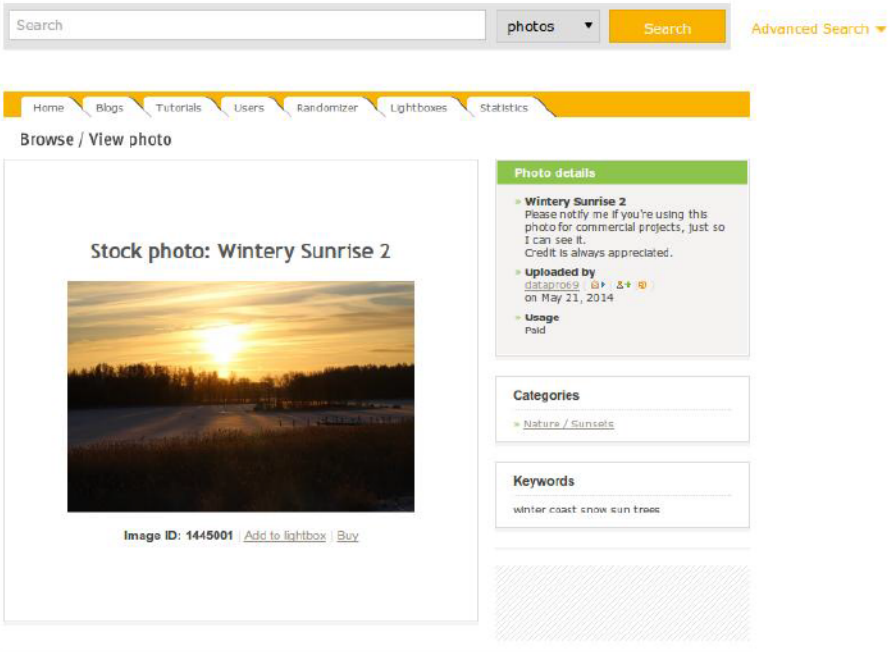
Los US implementados en el Sprint 1 permiten definir las funcionalidades básicas de la aplicación: la posibilidad de subir, catalogar y ordenar fotografías. Sin embargo, dado que uno de los objetivos principales de la misma es el hacer que estas fotografías sean de acceso público y/o que se pueda cobrar por su uso, es necesario extenderla para facilitar estas funcionalidades. En particular, se eligen del Sprint Backlog los siguientes US a atacar:

- **US5.** Como Usuario, debo poder buscar fotografías de mi interés utilizando un mecanismo de búsqueda textual.
- **US6.** Como Usuario, debo poder ver los detalles de una fotografía en particular y, de ser gratuita, descargarla.
- **US7.** Como usuario, debo poder comprar una fotografía si la misma no es de acceso gratuito.

Para implementar estas US, se construyeron los mockups HTML presentes en la Fig. 25.



Mockup 5 – Permite la búsqueda y visualización de fotografías (implementa US5)



Mockup 6 – Permite la visualización de una fotografía particular y su descarga o compra (implementa US6 y US7)

Figura 25. Mockups necesarios para la implementación del Sprint 2.

Las versiones taggeadas de estos mockups pueden observarse en la Fig. 26.

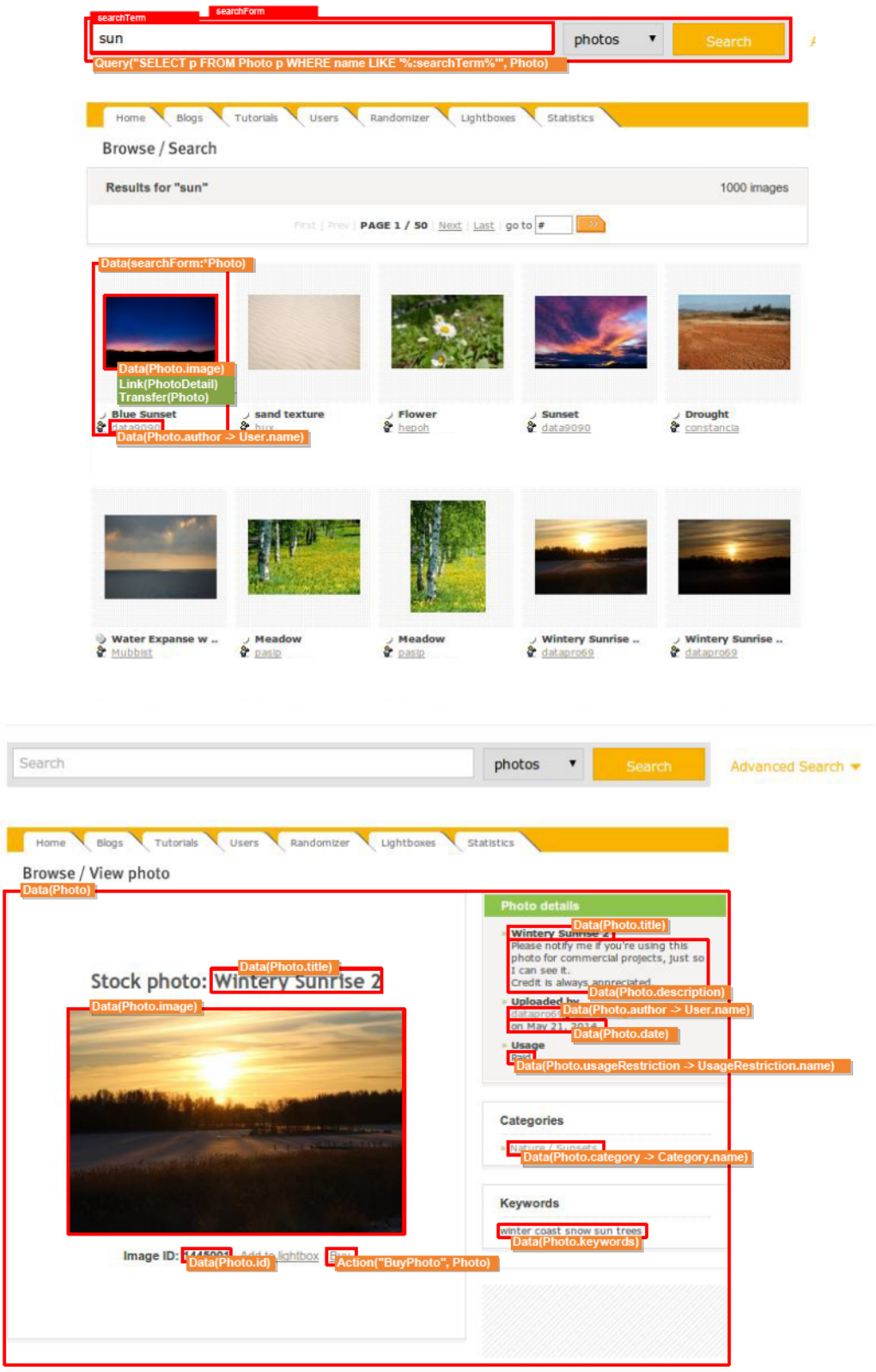


Figura 26. Mockups de la Fig. 25 taggeados, implementando lo planteado en las US del Sprint 2.

Varios casos de ejemplo de uso de los tags se han comentado ya la descripción del Sprint anterior. Los tags más relevantes a introducir en el primer mockup son:

- `Query("SELECT p FROM Photo p WHERE name LIKE '%:searchTerm%'", Photo)`. Denota que el `Panel` sobre el cual se aplica ejecutará la query descripta como primer parámetro al cargarse la pantalla. La ejecución de esta query devolverá una lista de fotografías (`Photo`), como se indica en el segundo parámetro del tag. En particular, se utilizará HQL[15] como lenguaje de consulta, dado que como motor de persistencia y mapping objeto-relacional se utilizó Hibernate[16]. En este contexto, se asume la existencia de tablas y clases con el mismo nombre que las clases capturadas desde los tags – en este caso, la clase `Photo`. La query hace referencia al parámetro `searchTerm`, el cual se obtiene desde la interfaz de usuario (a través del `InputBox` con ese id).
- `Data(searchFrom:*Photo)`. Denota que en la `Repetition` subyacente de mostrarán las fotografías obtenidas a través de la ejecución de la query especificada en el `Panel` `searchFrom`.
- `Link(PhotoDetail)` y `Transfer(Photo)`. Especifican que, al hacer click sobre la imagen de una `Photo`, se navegará hacia la pantalla `PhotoDetail` (Mockup 6 en la Fig. 25), enviando la `Photo` como parámetro.

El segundo mockup contiene varios tags que consisten en mostrar la información de la `Photo` asociada al panel principal a través del tag `Data(Photo)`. Esta fotografía es obtenida obligatoriamente a través de un parámetro, el cual es provisto en este caso por la pantalla anterior. Todas las propiedades – tags `Data(Photo.<propiedad>)` – ya han sido previamente introducidas en la sección anterior. El único tag novedoso en este contexto es `Action("BuyPhoto", Photo)`. El mismo denota que, al hacer un click en el `Link` *Buy*, se ejecutará una acción denominada `"BuyPhoto"`, la cual efectuará el pago de la fotografía. Dado que el pago será realizado a través de un servicio existente de pagos y no existe un tag para especificar este tipo de integraciones entre aplicaciones, la misma se representa como una acción abstracta. Al momento de generar la aplicación, se generará un fragmento de código el cual recibirá un objeto de tipo `Photo` y deberá ser completado a mano por los desarrolladores a fin de realizar la integración correspondiente.

Luego de aplicar los tags descriptos en la Fig. 26, la aplicación ya provee la posibilidad de buscar, visualizar y comprar fotografías – esta última, sólo en lo referente a interacción. Al observar los mockups de las figuras puede notarse que no todos sus elementos están taggeados. Esto no implica que los mismos nunca serán utilizados o que su comportamiento nunca será modelado sino que, aunque estos requerimientos visuales fueron descriptos por usuarios finales o clientes, los mismos no serán atacados en el Sprint actual.

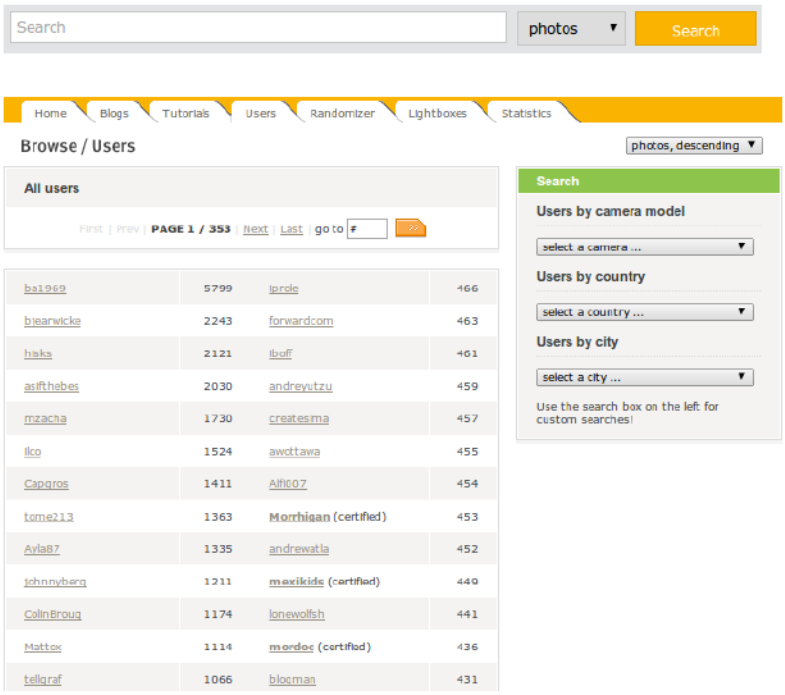
5.3. Sprint 3

Siguiendo las prácticas Scrum de MockupDD, durante los Sprints anteriores se modelaron e implementaron los requerimientos más prioritarios para los stakeholders: la posibilidad de publicar fotografías y cobrar por su uso. En este, que será el último Sprint que se incluirá en este caso de estudio, se

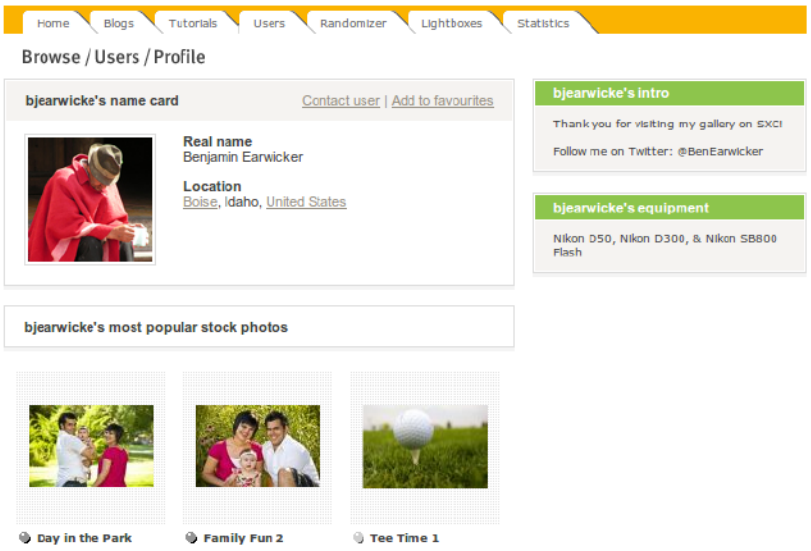
[15] HQL: The Hibernate Query Language – http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html
[16] Hibernate ORM – http://hibernate.org/orm

implementará la posibilidad de descubrir contenido navegando entre los usuarios de la aplicación. En particular, se considerarán las siguientes US:

- **US8.** Como Usuario, debo poder listar los diferentes usuarios que están registrados en la aplicación
- **US9.** Como Usuario, debo poder observar el perfil de un usuario particular, junto con sus fotografías publicadas



Mockup 7 – Permite el listado de usuarios en la aplicación (US8)



Mockup 8 – Muestra el perfil de un usuario particular (US9)

Figura 27. Mockups implementando las funcionalidades del Sprint 3.

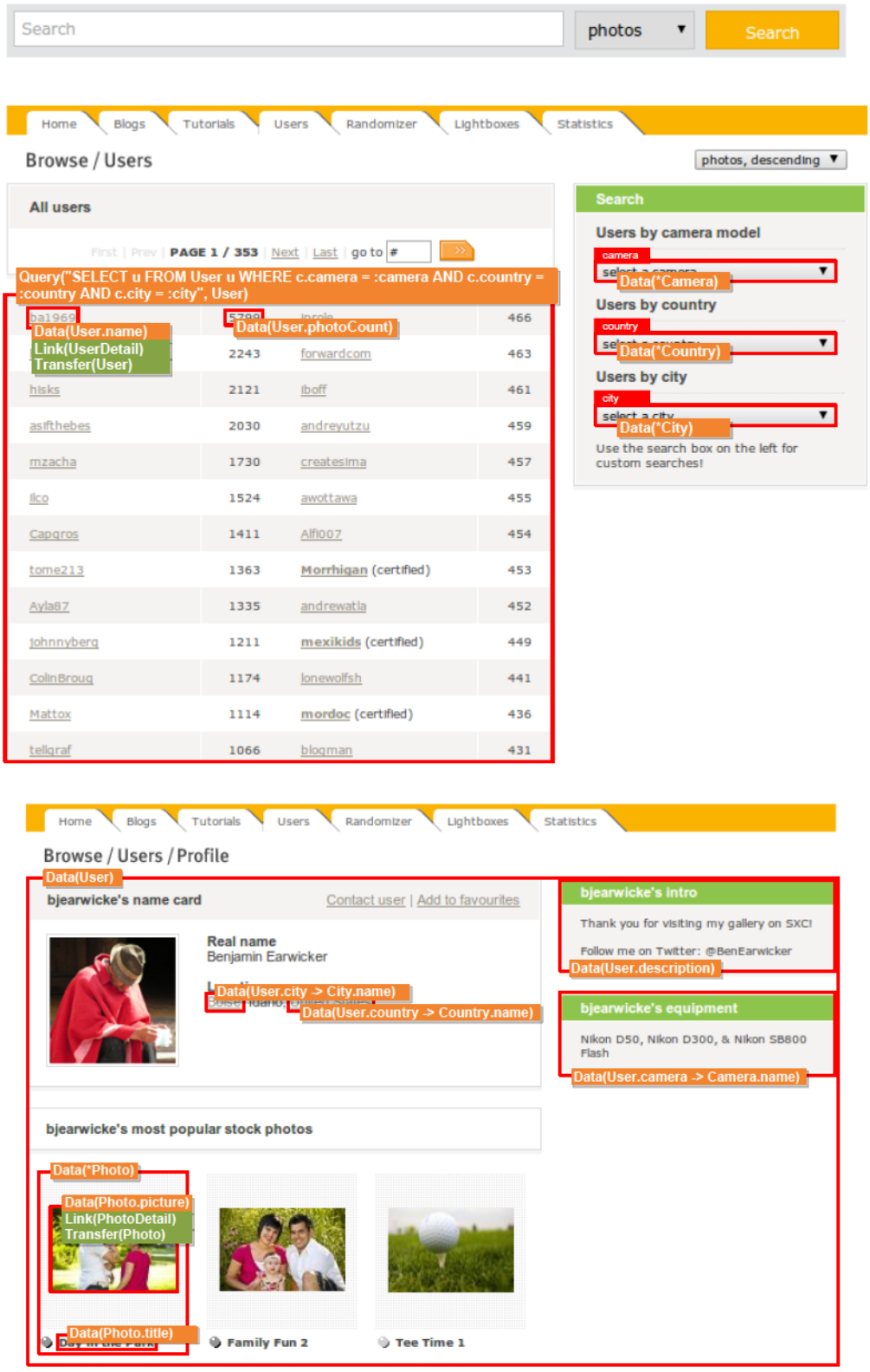


Figura 28. Mockups de la Fig. 26 taggeados, implementando lo planteado en las US del Sprint 2.

En la Fig. 27 se muestran los mockups que representan gráficamente las dos US de este Sprint, mientras que en la Fig. 28 se puede observar estos mismos mockups taggeados. Los tags utilizados en este Sprint no

son muy diferentes a los ya introducidos con antelación. El tag más complejo es que permite seleccionar las fotografías en base a los criterios de búsqueda introducidos: `Query("SELECT u FROM User u WHERE c.camera = :camera AND c.country = :country AND c.city = :city", User)`. La query representada por este tag toma los objetos de tipo `Camera`, `Country` y `City` (a partir de los `Dropdowns` nombrados `camera`, `country` y `city` respectivamente) y los utiliza para obtener las instancias de `User` relacionadas.

Los tags `Link(UserDetail)` y `Transfer(User)` permiten navegar hacia la segunda `Screen`, la cual muestra los detalles del usuario. Dentro de esos detalles, se pueden observar las fotografías. Los tags `Link(PhotoDetail)` y `Transfer(Photo)` permiten navegar hacia la misma `Screen` anteriormente ya introducida en el Mockup 6 de la Fig. 25. Habiéndose introducido ahora una vista para mostrar detalles de un `User`, se podría mejorar el Mockup 5 (Fig. 25, taggeado en la Fig. 26), permitiendo que al hacer click en el `User` se navegue hacia su perfil. Esto puede lograrse simplemente agregando los tags `Link(UserDetail)` y `Transfer(User)` sobre los links a los usuarios que en la Fig. 26 no están taggeados.

Cabe destacar que, a diferencia de los Sprints anteriores, en este Sprint se introdujo la clase `User` sin necesidad de haber planteado páginas las cuales permitan crear instancias nuevas de la misma. Sin embargo, en el contexto del proceso MockupDD Web, esto no impide que la aplicación pueda generase ni funcionar correctamente. Los artefactos de software subyacentes (tablas, clases, etc.) se generarán como se espera a partir de las relaciones inferidas desde los tags y los desarrolladores podrán cargar manualmente datos de prueba para poder mostrar la aplicación corriente a usuarios finales. Esto evita la necesidad de introducir más interfaces de usuario que las definidas para el Sprint, lo cual alargaría la iteración.

5.4. Discusión

En este capítulo se abordó la implementación de una aplicación real utilizando el proceso MockupDD adaptado desde Scrum. Se detallaron los primeros tres Sprints del desarrollo de una Aplicación Web de stock de fotografías clásica. Siguiendo el proceso esquematizado en la Fig. 16, en cada Sprint se definieron las User Stories (dando lugar al Sprint Backlog del Sprint), se construyeron los mockups necesarios para satisfacer estas US y luego se aplicaron los tags que permitieron enriquecer los mockups con comportamiento concreto.

Si bien los tags introducidos en cada uno de los Sprints han permitido describir los aspectos de interacción básicos, también se han dejado de lado aspectos no menos importantes los cuales no pueden ser expresados en MockupDD Web en su versión actual. Uno de ellos es la generación de propiedades/campos calculados o inicializados automáticamente. Un caso de ejemplo puede observarse en la Fig. 24, en el cual el tag `Data(Folder.date)` muestra la fecha de creación de un `Folder`. La misma no es ingresada por el usuario manualmente sino que es generada automáticamente en base a la fecha del sistema. En términos de código puntuales, introducir este cambio implicará agregar una línea en el constructor de la clase generada – por ejemplo, `this.date = new Date()`. Un caso similar puede observarse en la Fig. 28, en la cual la cantidad de fotografías de un usuario – plasmada por la propiedad `Data(User.photoCount)`. En este caso, el número de fotografías puede calcularse con una subquery (por ejemplo, `SELECT COUNT(*) FROM`

`Photo p WHERE p.user = :user`). Esta modificación implicará un cambio en la capa de datos de la aplicación generada o bien en la configuración del mapeador objeto-relacional – en el caso de la aplicación Photo Stock, se optará por esta última opción. Las acciones registradas a través del tag `Action()` – como el caso de `Action("BuyPhoto", Photo)` – serán generadas como fragmentos de código los cuales deben ser completados por el equipo de desarrollo. Estos detalles funcionales no bloquean la generación de demostraciones corrientes al final de cada Sprint. Estas versiones corrientes carecerán de estas funcionalidades puntuales, pero emularan por completo el resto de las características que la aplicación final implementará.

6. Evaluación

Para evaluar el enfoque MockupDD Web y sus herramientas de apoyo se realizaron diferentes experimentos controlados. En todos ellos se utilizó el método Goal-Question-Metric (GQM) [38] para conducir la evaluación. GQM representa un enfoque sistemático para la adaptación y la integración de las metas a los modelos de procesos de software, los productos y las perspectivas de interés de calidad, en base a las necesidades específicas de cada proyecto y organización. GQM define un determinado objetivo (*Goal*), refina ese objetivo con una serie de preguntas (*Questions*) y especifica las métricas (*Metrics*) que deberían proporcionar la información para brindar respuestas a estas preguntas. A continuación, se describirán cada una de las evaluaciones realizadas.

6.1. Evaluación general: MockupDD Web vs. MDWE tradicional

En esta evaluación se comparará la eficacia y eficiencia de la metodología MockupDD Web en comparación con una metodología MDWE tradicional. En este caso se ha elegido WebML [18] como metodología MDWE *de control* dado que la misma, además de ser una de las metodologías más populares, posee implementaciones funcionales las cuales se han probado en casos exitosos en la industria.

6.1.1. Objetivos

El modelo GQM sigue una estrategia *top-down*, comenzando con la definición de una meta explícita de medición. El propósito principal de esta evaluación es hacer una comparación cuantitativa del método MockupDD Web frente al un proceso de modelado tradicional MDWE *puro* para evaluar las mejoras que el enfoque puede ofrecer en términos de efectividad y eficiencia. La Tabla 2 muestra los objetivos generales de la evaluación, siguiendo el esquema propuesto por GQM.

Analizar	el producto Web desarrollado y su proceso de desarrollo
A los efectos de	comprender la metodología MockupDD Web
Con respecto a	la eficacia y la eficiencia del enfoque MockupDD Web
Desde el punto de vista	del equipo de proyecto
En el contexto de	un entorno de desarrollo Web con clientes y usuarios simulados

Tabla 2. El objetivo principal del experimento llevado cabo

6.1.1.1. Preguntas y Métricas

El experimento no incluye una evaluación de las ventajas del uso de mockups en el proceso de desarrollo, ya que esto ha sido evaluado y probado estadísticamente por Rica et al. en [8]. En esta evaluación, las siguientes preguntas de investigación fueron consideradas:

- **Pregunta 1:** ¿Proporcionan los tags provistos por MockupDD Web la semántica necesaria para representar y generar aplicaciones data-intensive con eficacia – es decir, de una manera precisa y completa?
- **Pregunta2**. ¿Permite MockupDD Web obtener modelos MDWE de manera más eficiente que construyendo estos modelos manualmente – es decir, con menos esfuerzo y menos incidencia de errores?

Con el fin de proporcionar evidencias relacionadas con estas preguntas, se definieron, recopilaron e interpretaron una cierta cantidad de métricas. Las métricas definidas se basan parcialmente en modelos WebML que se utilizaron para comprobar y comparar si el enfoque MockupDD Web tiene el mismo nivel expresivo y permite generar el mismo tipo de modelos, pero con más eficacia y eficiencia. Los participantes del experimento trabajaron con WebML y MockupDD Web con el fin de recopilar información cuantitativa de ambos enfoques. Las métricas definidas elegidas fueron las siguientes:

- **Métrica 1. Porcentaje de cobertura – Coverage Ratio (CR)**. Porcentaje de elementos del modelo WebML que pueden ser generados a partir de tags para especificar la semántica deseada. Proporciona una medida de la eficacia de la modelización de MockupDD Web.
- **Métrica 2. Tiempo Dedicado a la Corrección de Errores – Time Spent on Correcting Errores (TSCE).** Los errores son situaciones particulares en las que los participantes del experimento no han completado el modelado con éxito y por lo cual una nueva especificación debe ser considerada. El tiempo dedicado a la corrección de errores se calcula mediante el análisis de los modelos WebML generados a través de un tagging MockupDD Web y los construidos manualmente, comprobando errores puntuales (como elementos faltantes, semántica incorrecta, etc.). Ambos enfoques (MockupDD Web vs. WebML) se comparan para definir cuál generó modelos con menos errores. Esta métrica proporciona una medida de la eficiencia de modelado.
- **Métrica 3. Tasa de Completitud / Tiempo Medio dedicado al modelado - Completion Rate / Mean time on Task (CRMT)**. Esta métrica cuantifica el tiempo de modelado requerido en MockupDD Web y en WebML para generar la misma aplicación. Al igual que la métrica anterior, la misma proporciona una medida de eficiencia en el modelado.

Siguiendo el esquema de GQM, las Preguntas y Métricas fueron organizadas del siguiente modo: La Pregunta 1 se asoció a la Métrica 1 y la Pregunta 2 a las Métricas 2 y 3. De este modo, de ser aceptable un valor para la Métrica 1 la Pregunta 1 podrá responderse favorablemente y, de obtenerse valores razonables para las Métricas 2 y 3 se podrá responder positivamente la Pregunta 2.

6.1.2. Recolección de datos

Participantes. Para la recolección de la información estadística necesaria en la evaluación se organizaron 10 grupos académicos de desarrollo, cada uno compuesto por tres modeladores. Cinco grupos de participantes trabajaron usando MockupDD Web (grupos *Experimentales*) y otros cinco grupos utilizaron

WebML (grupos *de Control*), con su herramienta de soporte, WebRatio[17]. Los participantes fueron estudiantes avanzados de pregrado de Licenciatura en Informática / Sistemas y todos ellos estaban familiarizados con los métodos MDWE y, en particular, con WebML. Ambos grupos trabajaron en diferentes sesiones de laboratorio y fueron capaces de trabajar prácticamente día a día con los usuarios y clientes potenciales en los dos procesos. Los clientes y usuarios cuales fueron simulados por profesores pertenecientes a las carreras de Licenciatura en Informática y Sistemas de la Universidad de La Plata. En este contexto, se utilizó lenguaje natural, User Stories y mockups para refinar las especificaciones de requerimientos y también para expresar cualquier tipo de sugerencia adicional orientada a los participantes – es decir, los requerimientos fueron expresados en términos de Patrones de Interacción [39].

Mientras que los clientes y usuarios finales simulados estuvieron disponibles la misma cantidad de tiempo para ambos grupos (Control y Experimental), se observó que los participantes pertenecientes a los grupos de Control interactuaron de un modo más *espaciado* en comparación con los participantes de los grupos de experimentación. Esto último se atribuyó empíricamente al uso del proceso de trabajo tradicional *en cascada* de la metodología MDWE WebML.

Artefactos / Tecnología. Respecto a las herramientas y tecnologías utilizadas, para el caso de MockupDD Web se utilizaron todas las herramientas previamente expuestas (MHT, ITT, y DS), mientras que en el caso de WebML se utilizó su herramienta de soporte por defecto, WebRatio. WebRatio representa un entorno maduro, totalmente basado en modelos y se ha utilizado en varios proyectos a escala industrial[18], y su soporte técnico fue una de las razones que motivaron la elección de WebML como metodología de control para evaluar MockupDD Web. En términos generales, el enfoque de validación consistió en utilizar MockupDD Web para generar modelos WebML y comprobar si el proceso de modelado presentaba mejoras frente a la construcción manual de esos mismos modelos.

El Grupo Experimental aplicó la adaptación de Scrum planteada por MockupDD Web, dividiendo todo el desarrollo en tres Sprints de una semana de duración cada uno. En cada Sprint, un subconjunto de los mockups presentados se refinaron y taggearon. Por otro lado, los integrantes del Grupo de Control trabajaron siguiendo el workflow MDWE tradicional, sin un proceso estructurado en concreto – dado que WebML no lo define.

Procedimiento. La validación fue dividida en dos etapas diferentes. En ambas etapas se pidió a los participantes que construyan modelos para implementar el sitio Web de Photo Stock, introducido en detalle en la Sección 5. En ambos casos, se les proporcionó a todos los participantes un conjunto de mockups detallados y User Stories de la aplicación. El conjunto de mockups y User Stories fue la misma para todos los participantes en el experimento.

La primera etapa de la validación, pensada con el objetivo de responder a la Pregunta 1, consistió en la construcción manual de modelos WebML que implementen la aplicación solicitada y comprobando si MockupDD Web era capaz de expresar los mismos conceptos que estaban presentes en ellos utilizando tags, con el objetivo de evaluar su viabilidad semántica. En esta etapa, los participantes del Grupo de Control

[17] WebRatio – http://webratio.com
[18] WebRatio Success Stories – http://www.webratio.com/portal/content/en/success-stories,consultado el 07-May-2013

construyeron modelos WebML de la aplicación Photo Stock manualmente en base a los mockups y User Stories planteadas.

Para la segunda etapa, todos los participantes del Grupo Experimental recibieron una sesión inicial de entrenamiento en MockupDD Web en la cual un instructor explicó el enfoque – la creación de mockups, su tagging y las herramientas relacionadas. Más precisamente, las tareas realizadas fueron:

- Generar credenciales únicas para que cada participante pueda acceder a las herramientas de soporte de MockupDD Web.
- Instrucción en el proceso MockupDD Web (comentado el uso de conceptos tales como tags, SUI, etc.).
- Solicitarles a los participantes el desarrollo de la aplicación Photo Stock utilizando la metodología asignada (WebML o MockupDD Web dependiendo del grupo).
- Solicitarles a los participantes que mantengan seguimiento de los errores detectados durante ambos modelados, su tipo y el tiempo en las tareas de modelado en detalle.

Para aumentar la motivación por parte de los participantes, los instructores comentaron a los alumnos que el desarrollo Web a realizar en el experimento sería similar a los que lo deberían realizar para su trabajo de fin de curso.

Interpretación. El primer objetivo del experimento fue evaluar la eficacia de MockupDD Web calculando un valor para la Métrica 1 (CR). En la Fig. 29 se puede observar un resumen gráfico del Coverage Ratio en uno de que los modelos de WebML. En esa Figura, cada Data y Link Unit (conceptos WebML) son gráficamente cubiertos por su tag de origen (es decir, el tag que produce su creación durante el proceso de generación del modelo). Como se puede apreciar, los tags cubren casi la totalidad de los elementos. Adicionalmente, en algunos casos, un tag puede generar más de un elemento WebML (por ejemplo, un tag `Data()`, genera tanto un Data Unit como su Selector interno, el cual permite determinar la clase de objetos que se mostrarán en la misma. Asimismo, algunos elementos como Entry Units se generan a partir de elementos subyacentes de los mockups (por ejemplo, formularios de ingreso de datos). Dado que MockupDD Web no utiliza conexiones para expresar lógica de negocios o relaciones entre los componentes asociados, los Links entre las diferentes Units (por lo general entre 2 y 3 en promedio) se generan automáticamente en muchos casos a partir de un solo tag.

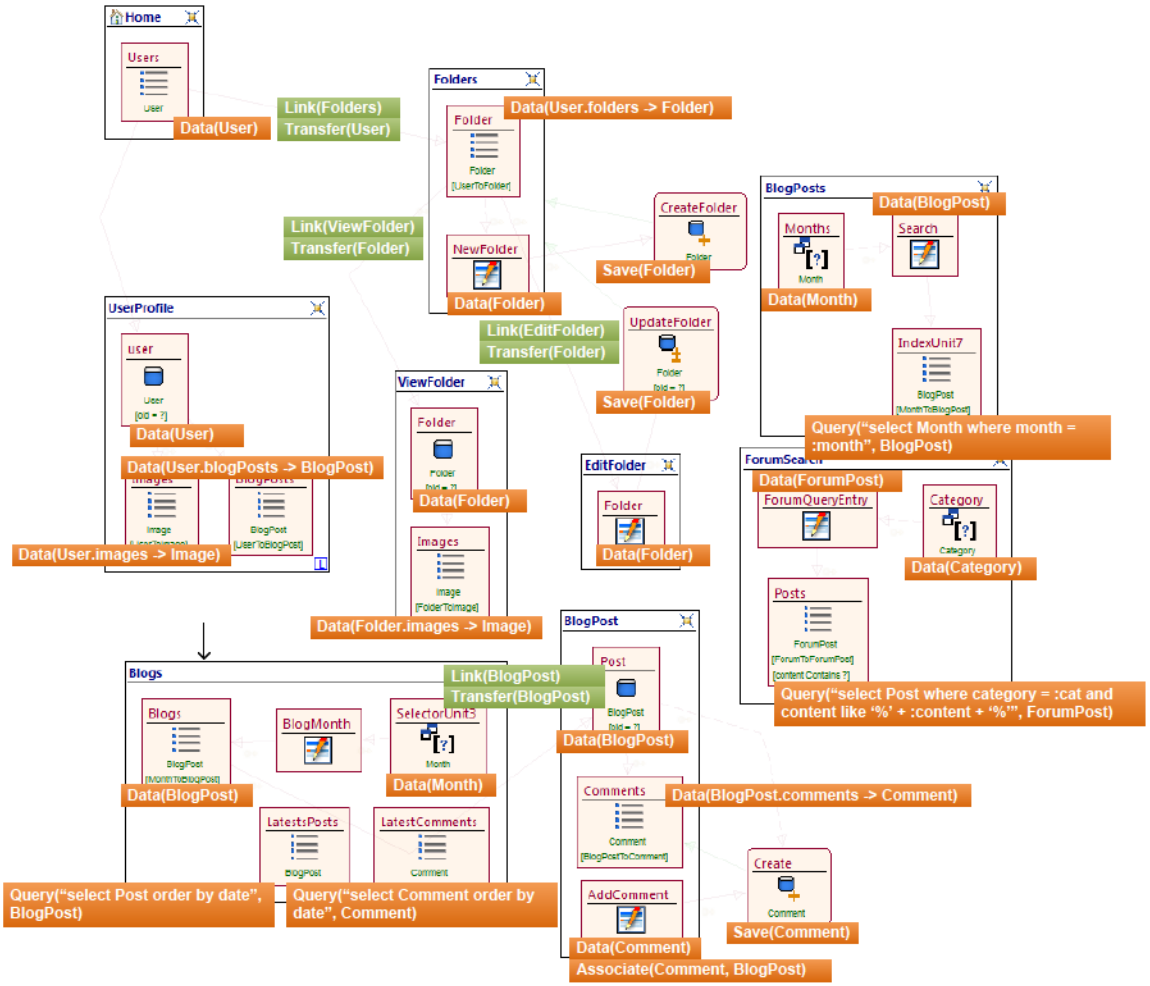


Figura 29. Cobertura de tags MockupDD Web sobre un modelo WebML de la Aplicación Web Photo Stock.

Los resultados del análisis detallado mostraron que el único elemento de modelo WebML que no se puede generar automáticamente desde tags fueron los selectores de las unidades en que se utilizó el tag `Query()`. Para cada modelo WebML construido por un participante se calculó la métrica CR – es decir, la proporción de elementos WebML modelados que se pudieron generar a partir de los tags MockupDD Web que cada participante ingresó. Los resultados mostraron que el CR promedio para los modelos WebML fue de entre 5,44% y 14,14% con una confianza del 98%. Además, como resultado del análisis se llegó a la conclusión de que los únicos elementos WebML que no pueden ser cubiertos por MockupDD Web son Query Units y ciertas funciones avanzadas como comportamiento AJAX. Debido a este resultado y el hecho de que las consultas ejecutables pueden expresarse libremente en MockupDD Web través de un tag `Query()` (por ejemplo, siendo escritas en lenguaje SQL convencional), puede concluirse que el enfoque MockupDD Web es eficaz en términos de la cobertura de tags, respondiendo así positivamente a la Pregunta 1.

Para computar un resultado para la Métrica 2 (TSCE), se confeccionó una escala de errores (descripta en la Tabla 3) la cual asigna un puntaje o *peso* a cada error posible. El puntaje es asignado a cada error de acuerdo al tiempo y esfuerzo estimado que lleva su corrección. Para los errores de elementos incorrectos o

faltantes, la puntuación asignada se basa en la cantidad de tiempo que se ahorra al omitir la creación de instancias de la Unit WebML *(Unit faltante),* de sus partes (*Atributo faltante o erróneo*) o de Links relacionados (*Link faltante*). El equipo de organización del experimento calculó y acordó estos valores proporcionalmente teniendo en cuenta el tiempo de modelado requerido en cada caso, a partir de los elementos menos costosos. Los cálculos se realizaron observando el comportamiento de modelado habitual de los participantes y en base a la experiencia personal del equipo. Finalmente, si bien la introducción de Units WebML extra no implica en general que algún requerimiento funcional no se cumpla en la aplicación, la misma añade tiempo de desarrollo el cual no es valuable directamente para los usuarios finales. De este modo, se retrasa así la puesta en prueba de la aplicación en pos de implementar un requerimiento que usuarios finales nunca solicitaron. En estos casos se decidió utilizar una puntuación similar a la de Units faltante pero, dado que el usuario final de todos modos obtendrá una aplicación final completa al final del proceso, sólo se computa la mitad del valor original de ese tipo de errores. Con este enfoque de puntuación, un modelo *perfecto* (es decir, idéntico al un modelo WebML *ideal*) tendrá una puntuación de cero, ya que no tiene que invertirse tiempo en corregirlo. Cualquier otro modelo que difiera del ideal tendrá una puntuación positiva, la cual indica la cantidad de tiempo proporcional necesario para corregir este modelo o la adición de componentes WebML que no cumplan requerimientos funcionales explícitos del usuario final.

Tipo de error	Puntaje	Descripción
Unidad adicional	0, 25	El modelador utiliza una Unit que puede ser omitida, lo que representa más esfuerzo de modelado
Unit faltante	0,5	Falta una Unit la cual implementa un requerimiento funcional valuable para usuarios finales
Unit más compleja	0,1	Se utilizó una Unit más compleja que la se podría haber usado para implementar determinada funcionalidad
Atributo faltante o erróneo	0,1	Un atributo de configuración de una Unit es erróneo o no fue especificado
Link faltante	0,1	Un Link que implementa un requerimiento funcional esperable por el usuario – por ejemplo, una navegación – no fue especificado.

Tabla 3. Escala de errores en el modelado.

La escala de errores mencionada fue utilizada para analizar el rendimiento de ambos grupos en cuanto a la calidad de los modelos WebML obtenidos. Con este objetivo, se definió un modelo WebML *ideal,* el cual se utilizó como referencia para comparar los modelos obtenidos en ambos casos (Control vs. Experimento) y analizar su calidad. Para este propósito, el grupo de organización del experimento definió cuidadosamente un llamado modelo *WebML de Control*, comprobando que cada concepto utilizado en el mismo fuera el más eficiente en términos de tiempo de modelado para expresar los requerimientos específicos de la aplicación. Dado que los miembros del equipo de organización del experimento tienen más de cinco años de experiencia en modelado WebML y aún más en el campo de MDWE, se asumió que el modelo obtenido es el más eficiente que se puede para satisfacer las exigencias de la aplicación de Photo Stock involucrada en el

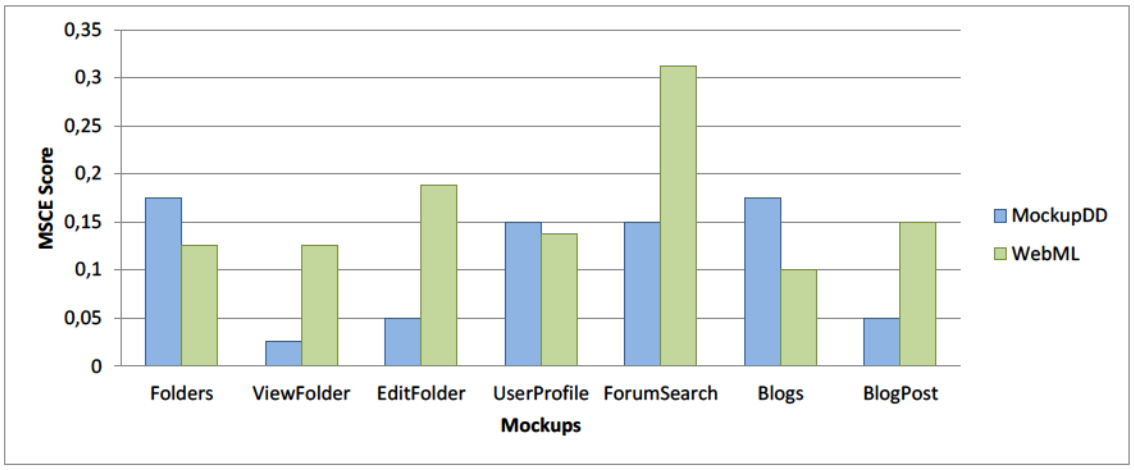
experimento. En este contexto, dado que los mockups restringen los widgets y la estructura de las páginas en el modelo WebML, la experiencia práctica ha demostrado que la mayor parte de las dudas de diseño en la construcción de estos modelos estuvieron relacionados con la elección de qué tipo específico de Unit que desea utilizar en cada caso (por ejemplo, Query vs Index Units, Multidata vs Index Units, etc.)

Usando el modelo de Control, se calculó Métrica 2 (TSCE) para cada uno de los modelos WebML construidos bajo cada metodología particular. Este cálculo implicó comparar los modelos construidos por los participantes y computar la sumatoria de todos los errores encontrados al compararlos contra el modelo WebML ideal. Luego, se utilizó la métrica Cliff's Delta [40] para medir el *tamaño del efecto* de la aplicación de MockupDD Web en el proceso de desarrollo. Se ha elegido utilizar este tipo de métrica debido a que permite el cálculo cuantificable entre dos grupos de manera no paramétrica y debido a que no requiere presunción de normalidad previa en los datos estadísticos. El resultado del cálculo del Cliff's Delta entre los modelos WebML construidos manualmente y aquellos generados a partir de mockups taggeados con MockupDD Web fue de -0,07, lo cual implica una leve mejora con respecto al tiempo en la corrección de errores a favor de MockupDD Web. Además de esta medida, se tomó un promedio de la métrica TSCE por mockup/página para analizar en qué casos MockupDD Web funcionó mejor que WebML y viceversa. Los resultados parciales están representados en la Fig. 30.a. En esta Figura también se muestra diagramas de tipo *box plot* con la distribución de la puntuación de error para WebML (Fig. 30.b) y MockupDD Web (Fig. 30.c) de acuerdo a las escalas definidas. Debido a esto, se puede concluir que el enfoque MockupDD Web no introdujo nuevos errores en el modelado en comparación con WebML manual, y también mostró una leve mejora en términos de definición de modelos correctos desde el principio del desarrollo. Como dato adicional, se observó que los participantes que construyeron modelos directamente con WebML fueron más propensos a cometer errores con una puntuación más alta, mientras que los que utiliza MockupDD Web tendían a cometer errores de menor valor.

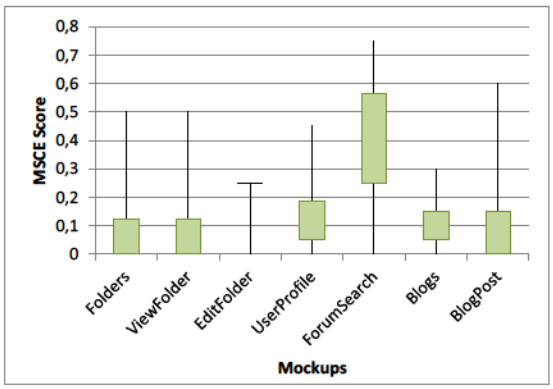
Para la Métrica 3 (CRMT), se evaluó el tiempo insumido por todos los participantes durante las tareas de modelado; es decir, la instrucción inicial y el tagging en el caso MockupDD Web, y la construcción directa del modelo en el contexto de WebML. Se les solicitó a los participantes tomar cuidadosamente el tiempo transcurrido en el modelado como un requerimiento clave en su trabajo. Al igual que antes, se calculó el Cliff's Delta para medir el tamaño del efecto de la aplicación de MockupDD Web en comparación con WebML, esta vez en relación con el tiempo necesario para obtener los modelos. El cómputo tuvo como resultado -0.89, lo que implica que las medidas de tiempo de desarrollo de MockupDD Web fueron significativamente menores en promedio que en el caso WebML. En la Fig. 31 se puede observar gráficamente representado el tiempo promedio tomado para las diferentes actividades en el contexto de cada metodología particular.

Habiendo logrado resultados positivos para las Métricas 2 y 3 (TSCE y CRMT respectivamente), podemos concluir que la Pregunta 2 también puede responderse positivamente. Lejos de ser ideal (como puede verse, por ejemplo, el análisis de la cantidad de errores y el tiempo empleado en la corrección de ellos se encuentran en la Fig. 30), luego de haber respondido positivamente a las Preguntas definidas a través de las Métricas detalladas, podemos concluir que en términos generales el enfoque MockupDD Web ofrece una

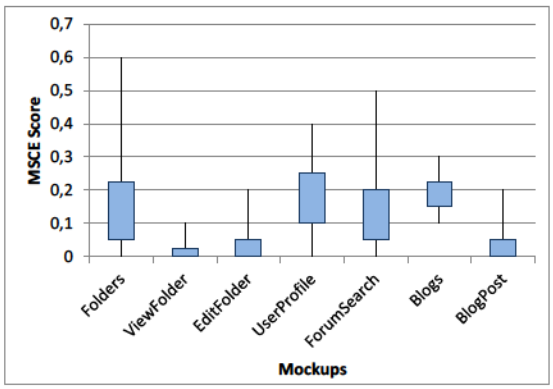
estrategia de construcción de modelos más rápido y menos propenso a errores, en comparación con las metodologías MDWE clásicas.



(a) Resumen de la métrica TSCE computada por mockup y enfoque



(b) Puntaje TSCE por mockup (WebML)



(c) Puntaje TSCE por mockup (MockupDD)

Figura 30. Resultados de la métrica TSCE para WebML y MockupDD Web.

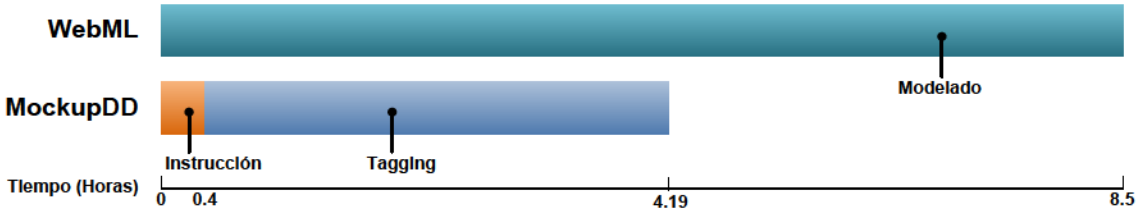


Figura 31. Distribución promedio de tiempo en los enfoques WebML y MockupDD Web.

6.1.3. Amenazas a la validez y limitaciones

Aunque efectiva, la evaluación llevada a cabo también presenta ciertas amenazas a su validez. Las amenazas más importantes que percibió el equipo de organización del experimento fueron:

1. **Conjunto de pruebas reducido**. La validación de MockupDD Web fue realizada sólo con una aplicación data-intensive. Aunque las aplicaciones de este tipo son similares entre sí en cuanto a sus requerimientos estructurales y de comportamiento, la aplicación particular considerada pudo haber favorecido a MockupDD Web en los resultados en comparación a WebML de algún modo puntual. Para aminorar esta amenaza, se analizaron varios ejemplos clásicos y bien conocidos de sitios Web data-intensive (por ejemplo, los que se incluyen con la herramienta WebRatio) y se eligió uno similar en tamaño y complejidad.
2. **Experiencia industrial de los participantes**. Si bien al momento de realizar el experimento los participantes del experimento mostraron conocimientos necesarios desarrollo Web y en el área de MDWE, en general no se caracterizaron por una experiencia industrial a largo plazo y comprobable. El haber contado con participantes con mayor experiencia comprobable en proyectos de desarrollo de software permitiría haber obtenidos resultados más precisos respecto a la aplicabilidad de MockupDD Web en la industria. Para mitigar el impacto de esta amenaza, se seleccionaron los estudiantes que participaron en el desarrollo de al menos dos de Aplicaciones Web en el contexto de su carrera de grado.
3. **Familiaridad con el dominio**. Los ejemplos de aplicaciones data-intensive son bastante reconocidos (por ejemplo, sitios de e-commerce o blogs). Los participantes del experimento pudieron haber tenido un conocimiento inicial del dominio y de la lógica de negocio que puede haber influido en el modelado de aplicaciones bajo ambos enfoques comparados en la validación. Para evitar ese problema, se les consultó a los alumnos acerca del dominio de la aplicación Photo Stock utilizada para asegurar que el mismo fue totalmente nuevo para ellos.
4. **Modelo WebML de Control.** El modelo WebML de Control utilizado para calcular algunos de las métricas del experimento de validación y que fue considerado como *ideal*, pudo haber contenido características que puedan ser modeladas aún más rápido o más eficientemente. Si se encontrara un modelo más simple que satisfaga los mismos requerimientos de la aplicación Photo Stock, entonces la métrica TSCE calculada para los modelos construidos manualmente frente a los generados por MockupDD Web podría haber resultado desfavorable para esta última. Para reducir esta amenaza potencial, el modelo WebML de Control fue construido con cuidado por un equipo de profesionales con al menos cinco años de experiencia en el lenguaje WebML y aún más en el área de MDWE.
5. **Escala de errores.** Como fue comentado con antelación, la métrica de TSCE se calcula en base a una escala de errores puntual, en donde a cada tipo de error se le asigna un puntaje de acuerdo al tiempo relativo que lleva su corrección. Por ende, errores en esta escala puede dar lugar a fallos en la evaluación de los modelos (por ejemplo, percibiendo erróneamente que aquellas fallas más frecuentes en los modelos MockupDD Web tuvieron menos valor que aquellas encontradas en los modelos construidos manualmente). Para mitigar esta amenaza, el equipo de organización del experimento volvió a comprobar para todos los modelos, que la corrección de errores realizados en el Grupo de Control efectivamente tomarían más tiempo para corregirse que los cometidos por

en comparación con el Grupo Experimental, luego de haber llevado a cabo la ejecución de las sesiones experimentales.

6.1.4. Lecciones Aprendidas

En el experimento se han cuantificado y validado los beneficios de MockupDD Web en lo relativo al modelado de Aplicaciones Web. Sin embargo, no se han incluído hasta el momento detalles acerca de cómo los participantes se comportaron en el contexto del proceso ágil y MDD que propone este enfoque. En este apartado se comentarán diferentes situaciones observadas en la práctica durante las sesiones de modelado con MockupDD Web. Las mismas fueron comparadas con los comportamientos de modelado observados durante las sesiones de trabajo con WebML. La principal intención de esta sección es la de capturar parte de la experiencia que no se ha podido cuantificar concretamente en relación con las características y prácticas ágiles observadas en el proceso. Durante las sesiones de modelado con MockupDD Web se observaron las siguientes prácticas:

1. **Aprendizaje iterativo del lenguaje de modelado.** Como se ha mencionado anteriormente, los participantes del experimento ya poseían cierta experiencia en metodologías MDWE y, en particular, en WebML y WebRatio. Mientras que los mismos aprendieron y practicaron WebML con WebRatio a través de cursos específicos que fueron de dos a tres meses de duración, MockupDD Web se introdujo con una sesión de instrucción durante menos de una hora. Luego, los modeladores aprendieron los aspectos restantes durante el modelado, utilizando las metáforas de interfaz de usuario presentes en mockups para comprender la semántica y las reglas de las construcciones del lenguaje (representadas a través de tags). El equipo de organización del experimento, compuesto mayormente por docentes en el área de MDWE, pudieron comprobar de este modo que el enfoque de modelado MockupDD Web resulta más sencillo y rápido para ser aprendido sobre la marcha, iterativamente.
2. **Mayor frecuencia de retroalimentación entre desarrolladores y usuarios finales**. Dado que MockupDD Web permite generar una demostración de la aplicación siendo modelada al instante, la generación de prototipos funcionales para ser mostrados a clientes o usuarios finales fue sensiblemente más frecuente que en WebML. En este punto, es importante señalar que WebRatio, como toda herramienta de soporte a una metodología MDWE, ya provee sus propios mecanismos para la generación de aplicaciones funcionales desde los modelos construidos hasta el momento. Sin embargo, la generación de aplicaciones funcionales en este caso requiere más esfuerzo que en MockupDD Web – implicando la generación, compilación, deployment y reinicio del servidor de pruebas locales. Debido a esto, la muestra de prototipos funcionales a usuario finales (simulados por el equipo de organización del experimento) fueron menos frecuentes en el caso de WebRatio/WebML. En muchos casos, los miembros del equipo de organización experimento fueron capaces de corregir errores de interpretación en los requerimientos tan pronto como fueron percibidos en la versión de demostración de la aplicación. De este modo, la experiencia demostró que en lugar de discutir los requerimientos de forma oral o textual, la propia aplicación se

utiliza más a menudo para hablar de la versión de desarrollo actual en comparación con el caso WebML. Al mismo tiempo, la interacción con fluidez mostró claramente quienes eran los desarrolladores más activos en los grupos en el contexto de MockupDD Web.

3. **Dudas más explícitas por parte de los desarrolladores.** En el punto anterior, las demostraciones funcionales de la aplicación funcionaron como un origen *pasivo* para los desarrolladores en pos de detectar potenciales errores en los requerimientos. Sin embargo, se notó durante el experimento que MockupDD Web fomentó preguntas directas de parte de los usuarios más frecuentemente que en el caso de WebML, utilizando las metáforas visuales presentes en los mockups.
4. **Modelado de la aplicación de un modo más general.** Finalmente, en el caso de MockupDD Web se notó que los desarrolladores tendieron a modelar las aplicaciones de un modo más *horizontal*, en iteraciones cortas, abarcando diferentes funcionalidades presentes en diversas pantallas (mockups) en comparación con WebML. En este último, los modeladores avanzaron *en profundidad*, esperando a terminar y refinar cada funcionalidad puntual en cada pantalla hasta avanzar con la siguiente.

En base a las características observadas, los puntos (2) y (3) han ayudado a establecer empíricamente que MockupDD Web cumple con algunas características ágiles con más éxito que las metodologías MDWE clásicas como WebML. En general, y puntualmente teniendo en cuenta los principios comentados en el Agile Manifesto[19], se halló que el principio de *[...] promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely* es más fácil de cumplir con MockupDD Web que utilizando metodologías como WebML. Esto se debe a que a las mejoras en términos de frecuencia y calidad de interacción entre desarrolladores y usuarios finales que la MockupDD Web introduce en comparación a WebML. Debido a que se evaluó sólo una aplicación en detalle y se utilizó (a través de generación automática de modelos) un solo conjunto de tecnologías y arquitectura, no fue posible comprobar principios relacionados con el trabajo de equipo en el desarrollo en períodos más largos de tiempo como *Continuous attention to technical excellence and good design enhances agility*, *The best architectures, requirements, and designs emerge from self-organizing teams* or *at regular intervals, the team reflects on how to become more effective [...]*.

6.2. Evaluación de modelado de datos: MockupDD Web vs. Modelado tradicional

Dado que la metodología MockupDD Web está orientada a la definición de Aplicaciones Web data-intensive, la construcción de un modelo de datos correcto y completo representa una tarea esencial en el proceso de desarrollo. Por este motivo y en base a los resultados observados en la evaluación anterior, se llevó a cabo una evaluación adicional de MockupDD Web específicamente orientada al modelado de datos. A través de la misma se comprobó que MockupDD Web permite construir modelos de datos en base a tags de modo más eficiente en comparación a utilizar mockups y User Stories sólo como artefactos de requerimientos en su estado puro y construyéndolos manualmente en base a estos últimos.

[19] Principles Behind the Agile Manifesto – http://agilemanifesto.org/principles.html

El esquema de validación en este caso en particular fue similar a la validación descripta en la subsección anterior, utilizando nuevamente la estrategia GQM.

6.2.1. Objetivos

Los objetivos de esta validación (ver Tabla 4) se describen utilizando una tabla similar a la ya introducida Tabla 2. El objetivo general será hacer una evaluación similar a la planteada anteriormente, pero sólo considerando la modelización de datos.

Analizar	los modelos de datos obtenidos
A los efectos de	evaluar MockupDD Web en el contexto del modelado de datos
Con respecto a	su productividad, incidencia de errores y trazabilidad de requerimientos
Desde el punto de vista	del equipo de proyecto
En el contexto de	un entorno de desarrollo Web con clientes y usuarios simulados, abarcando sólo la etapa de modelado de datos

Tabla 4. El objetivo principal del experimento de modelado de datos llevado a cabo.

6.2.2. Preguntas y Métricas

Para esta evaluación, se definieron las siguientes Preguntas:

- **Pregunta 1**: ¿Permite MockupDD Web construir modelos de datos más rápidamente utilizando tags en comparación a su definición manual?
- **Pregunta 2**: ¿Permite MockupDD Web construir modelos de datos más precisos (es decir, con menos elementos faltantes, mal definidos y no explícitamente requeridos) en comparación a aquellos construidos manualmente?
- **Pregunta 3**: ¿Permite MockupDD Web construir modelos de datos más trazables desde el punto de vista de los requerimientos de la aplicación siendo modelada?

En los tres casos, *manual* o *manualmente* hace referencia al uso de mockups y User Stories sólo como artefactos de especificación de requerimientos.

Para responder estas tres preguntas, se plantearon 3 métricas diferentes:

- **Métrica 1. Tiempo de Construcción – Building Time (BT)**: Tiempo que lleva construir el modelo de datos de la aplicación utilizando una estrategia particular (MockupDD Web o *manual*).
- **Métrica 2. Puntaje de Errores – Error Score (ES)**: Un puntaje numérico indicando el impacto de los errores encontrados en los modelos de datos construidos utilizando una estrategia particular. Al igual en la validación anterior, se recurre a un *Modelo de Control* construido por expertos como referencia para comparar cada modelo generado y evaluar cuánto difieren de ese modelo *ideal*.
- **Métrica 3. Elementos no Trazables – Non-Traceable Elements (NTE)**: La cantidad de elementos no trazables que fueron encontrados en cada modelo de datos construido bajo una estrategia particular – es decir, la cantidad de elementos en esos modelos los cuales no pueden

asociarse a ningún elemento de los artefactos de requerimientos principales utilizados – mockups.

Para calcular la métrica de calidad en esta validación (ES) se utilizó una estrategia similar a la usada para la calcular la métrica TSCE en la validación anterior: se comparó cada uno de los modelos construidos por los participantes contra un Modelo de Control. Las diferencias fueron pesadas de acuerdo a una escala de errores la cual se describe en la Tabla 5. Nuevamente, los artefactos requeridos para computar la métrica de calidad ES (los Modelos de Control para cada aplicación y la escala de errores) fueron especificados y diseñados por un equipo de profesores de la Universidad Nacional de La Plata con más de 7 años de experiencia en modelado de datos y orientación a objetos. Los modelos fueron construidos por este grupo de expertos sin asistencia de ninguna herramienta para evitar cualquier tipo de *bias* introducido por las mismas en el modelado.

Tipo de error	Puntaje	Descripción
Clase faltante	1	Falta una clase requerida en el modelo
Atributo faltante	0,5	Falta un atributo requerido en una clase del modelo
Asociación faltante	0,5	Falta una asociación requerida entre dos clases del modelo
Herencia faltante	0,5	Falta una relación de herencia requerida entre dos clases del modelo
Clase extra	0,5	Se ha definido una clase extra la cual no ha sido explícitamente definida en los artefactos de requerimientos
Tipo de dato incorrecto	0,25	Un atributo en una clase tiene un tipo de dato incorrecto
Atributo/asociación extra	0,25	Se ha definido un atributo o asociación sin haber especificado explícitamente en los artefactos de requerimientos utilizados

Tabla 5. Diferentes tipos de errores de modelado de datos y su *puntaje de error* asignado.

Siguiendo el esquema GQM, las preguntas y métricas definidas fueron organizadas del siguiente modo: la Métrica 1 permitirá contestar la Pregunta 1, la Métrica 2 proveerá indicios para contestar la Pregunta 2 y finalmente la Métrica 3 permitirá obtener una respuesta para la Pregunta 3.

6.2.3. Recolección de datos

Participantes. Para reunir información estadística, fueron reclutados 30 estudiantes de Licenciatura en Informática y Licenciatura en Sistemas de la Facultad de Informática dependiente de la Universidad Nacional de La Plata. Antes del ejecutar el experimento controlado, cada participante fue sometido a una entrevista para asegurar que posea los conocimientos básicos necesarios de modelado de datos y Orientación a Objetos. El modelado de datos manual se llevó a cabo utilizando modelos OO escritos en el lenguaje Java, dado que tanto el paradigma OO como el lenguaje son muy populares entre los estudiantes. Por este motivo, antes del ejecutar el experimento se comprobó que todos ellos tuvieran conocimientos básicos de le lenguaje Java y de Orientación a Objetos. Como se describirá luego, cada participante trabajó con ambos enfoques (modelado manual y MockupDD Web). El grupo de estudiantes estuvo compuesto por 18 hombres y 12 mujeres, con un rango de edades de entre 22 y 30 (media 26,3), poseyendo de 2 a 4 años (media 2,3) de

experiencia en modelado de datos. Se controló también que todos ellos estuvieran familiarizados como IDEs reconocidas para el lenguaje Java (19 de ellos reportaron haber usado Eclipse[20] alguna vez y 11 de ellos NetBeans[21]). Esto último resulta importante debido a que el uso de IDEs agiliza a través de características como refactorings y asistencia de código a escribir modelos de OO más rápidamente, llevando a una comparación más justa entre ambas metodologías. Finalmente, 27 de los estudiantes manifestaron tener conocimiento en manejo de bases de datos relacionales y uso de sus herramientas asociadas.

Artefactos/Tecnología. En el caso de modelado manual, se brindó a los participantes la posibilidad de usar su IDE de preferencia para construir modelos manualmente, siempre y cuando la misma provea la asistencia y facilidades antes descriptas a la hora de modelar. Se brindó una capacitación inicial para instruir en el uso de todas las características de las IDEs elegidas, en pos de hacer que la construcción de modelos bajo ambas metodologías cuente con una asistencia técnica similar. Para el modelado con MockupDD Web, los participantes utilizaron las mismas herramientas que fueron introducidas con antelación.

Procedimiento. A cada participante se le brindó un conjunto de 3 conjuntos de mockups pertenecientes a 3 aplicaciones diferentes – las cuales serán llamadas Aplicación A, B y C. Para estimular el desarrollo en el contexto de Aplicaciones Web modernas, el equipo de organización del experimento definió estas aplicaciones en base a websites existentes – por ejemplo, portales de música online o de preguntas y respuestas. Se proveyeron un conjunto de 5 mockups por aplicación. Cada participante debió construir modelos de datos tomando estos mockups como artefacto de requerimientos principal. El modelado fue dividido en dos iteraciones por aplicación. Para obtener la misma cantidad de muestras por aplicación y estrategia de modelado, de los 3 modelos a construir por cada participante, 2 fueron construidos con una de las estrategias (manual o MockupDD Web) y la otra con la estrategia restante. El equipo de organización del experimento prestó especial atención a registrar el tiempo insumido por cada uno de los participantes para construir cada uno de los modelos. Las estrategias de modelado y mockups fueron distribuidas por los participantes de modo de asegurar este balance y, además, evitar el potencial bias introducido por que un participante modele una aplicación con una de las estrategias y luego con la otra – lo que se conoce como una amenaza de *maduración* a la validez del experimento [41]. Finalmente, el orden en que se modelaron las diferentes aplicaciones fue aleatorio. Para considerar un contexto de diseño participativo, parte de los profesores del equipo de organización del experimento actuaron como clientes y usuarios finales, solventando posibles dudas y ambigüedades presentes en los mockups sin utilizar jerga técnica – es decir, sólo expresándose en términos de objetos de negocios. Estos profesores estuvieron también involucrados en la construcción de los mockups y en la etapa de tagging. Sin embargo, no estuvieron involucrados en la elaboración del enfoque MockupDD Web ni estuvieron presentes durante las sesiones de capacitación en la metodología. Finalmente, con el fin de obtener datos estadísticos que permitan cuantificar las métricas planteadas con antelación, se les solicitó a los participantes llevar una cuenta minuciosa del tiempo invertido en la construcción de los modelos bajo cada estrategia.

[20] Eclipse – https://www.eclipse.org/
[21] NetBeans – https://netbeans.org/

Interpretación. Para medir el tamaño del efecto de la aplicación de MockupDD Web en el proceso de modelado en comparación al modelado de datos tradicional utilizando mockups, se utilizó la métrica Cliff's Delta como en la validación anterior.

Para la métrica BT se computó el Delta por aplicación y metodología utilizando las métricas de tiempo registradas por el equipo de organización del experimento. En la Fig. 32 se muestran dos box plots que resumen la distribución de tiempo de modelado en cada estrategia por aplicación. El resultado de este cómputo arrojó los valores -0,45, -0,48 y -0,9 para las aplicaciones A, B y C respectivamente. De acuerdo a las semántica del Cliff's Delta, estas tres métricas permiten asegurar con firmeza que el modelado con MockupDD Web llevó efectivamente menos tiempo en comparación a la construcción manual de los modelos de datos.

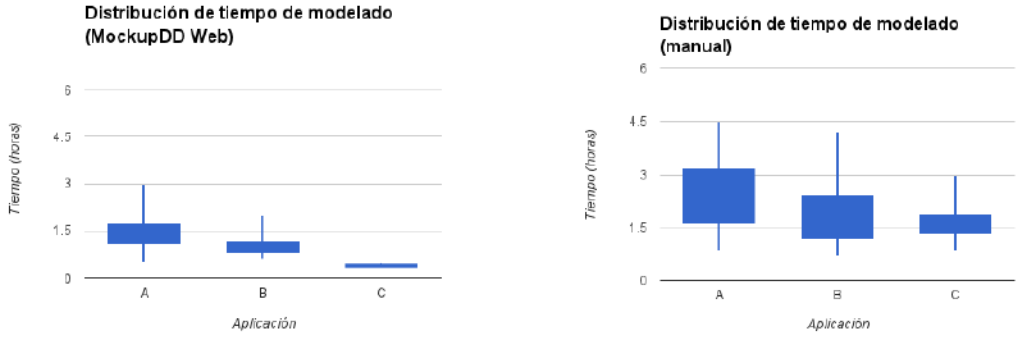


Figura 32. Distribución del tiempo de modelado por aplicación y estrategia

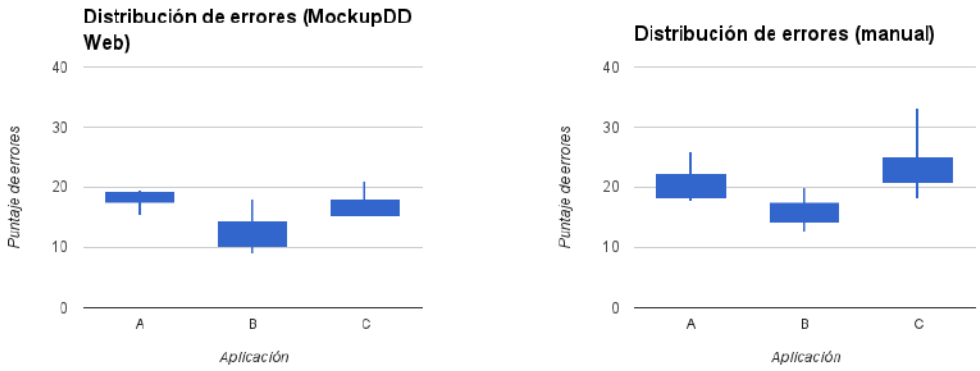


Figura 33. Distribución de puntaje de errores por aplicación y estrategia

En el caso de la métrica ES, en base a la escala de errores antes definida, se computó un puntaje de error para cada aplicación y participante comparando el modelo generado bajo la metodología asignada con el modelo de datos de Control diseñado. El puntaje de errores por aplicación y participante consistió en la suma de los errores individuales que se encontraron a través la comparación. En la Fig. 33 se muestran dos diagramas de cajas en los cuales se grafica la distribución de puntaje de error por aplicación y participante. De modo similar a la métrica BT, se computo el Delta por aplicación para obtener el tamaño de efecto de aplicar MockupDD Web al proceso de modelado, en este caso en lo relativo a errores en los modelos. El resultado del cómputo arrojó los valores -0,12, -0,52 y -0,87 para las aplicaciones A, B y C respectivamente.

Nuevamente, de acuerdo a las semántica del Cliff's Delta, a través de estas métricas puede responderse la Pregunta 2 afirmativamente y, de este modo, afirmar que MockupDD Web permite construir modelos de datos con menor cantidad de errores en comparación a su definición manual.

Una estrategia similar se utilizó para cuantificar la métrica NTE (cuya distribución se muestra en la Fig. 34. El resultado de este cómputo arrojó los valores -0,46, -0,7 y -0,53 para las aplicaciones A, B y C respectivamente. Nuevamente, esta métrica permitió responder la Pregunta 3 positivamente, dado que los modelos de datos construidos con MockupDD Web mostraron una notoria mejora en lo relativo a trazabilidad en comparación aquellos modelos construidos manualmente.

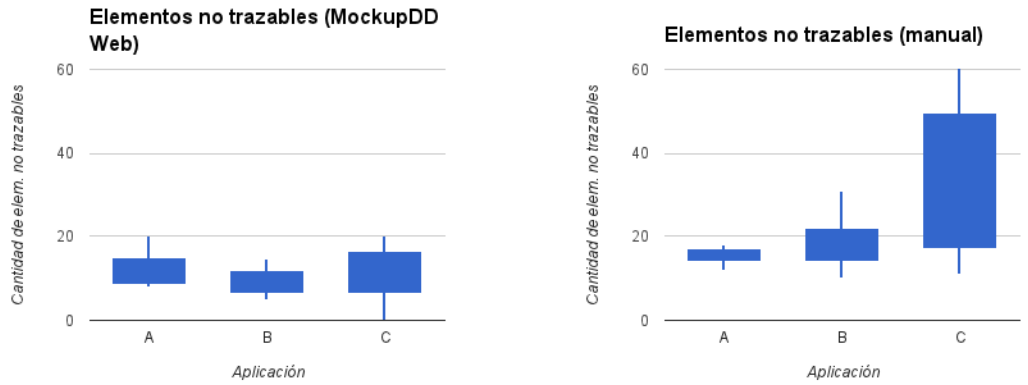


Figura 34. Distribución de elementos no trazables en modelos generados utilizando MockupDD Web vs. modelos construidos manualmente.

Con el objetivo de reforzar los resultados positivos obtenidos por la métrica de Cliff's Delta, se llevó a cabo un test-t para cada una de las métricas computadas. En este caso, se unieron las 3 muestras obtenidas por aplicación en una sola, formando dos muestras por metodología – manual vs. MockupDD Web. Dado que los tamaños de muestra por aplicación fueron los mismos para ambas metodologías, se asume que unir los mismos en una sola muestra por cada una no representa una amenaza estadística al test – debido a que cada aplicación tiene el mismo *peso* en la muestra general en ambos casos. Los resultados fueron los siguientes:

- Para la métrica BT se concluyó que, en base a las muestras obtenidas, la media de tiempo de construcción de modelos en el caso de MockupDD Web es menor a la media de tiempo de construcción manual con una confianza del 99%.
- Para la métrica ES se concluyó que, en base a las muestras obtenidas, la media de puntaje de error en el caso de MockupDD Web fue menor que la media en el caso de modelado manual con una confianza de 95%.
- Para la métrica NTE se concluyó que, en base a las muestras obtenidas, la media de elementos no trazables en el caso de MockupDD Web fue menor que la media de la muestra al modelar manualmente, con una confianza del 95%.

Los resultados de los 3 tests estadísticos reafirmaron las repuestas positivas a las 3 Preguntas planteadas en la evaluación.

6.2.4. Amenazas a la validez

En la evaluación descripta se constató a través de un experimento controlado y siguiendo el esquema GQM que MockupDD Web es significativamente más eficiente en términos de tiempo y de calidad para construir modelos de datos en comparación a su construcción manual. En ambos casos se utilizaron mockups como artefactos de requerimientos, con al salvedad de que en MockupDD Web pudieron ser aprovechados directamente para permitir la construcción iterativa de modelos de datos utilizándolos como base.

Sin embargo, el equipo de organización y ejecución del experimento encontró y trató de mitigar diferentes amenazadas a la validez del experimento, a saber:

- **Errores en el Modelo de Control**. El Modelo de Control es un artefacto esencial para computar la métrica ES. Debido a esto, errores en este modelo pueden comprometer los valores obtenidos para esta métrica. Para mitigar esta amenaza y al igual que ocurrió con el Modelo de Control en la validación anterior, este modelo fue construido por miembros del equipo de organización del experimento. Éstos poseen más de 7 años de experiencia en el área de modelado de datos en aplicaciones reales.
- **Falta de experiencia de los modeladores**. La falta de experiencia de los modeladores que participaron del experimento pueden conllevar a resultados no uniformes – por ejemplo, aquellos menos experimentados pueden demorar más en construir ciertas partes de los modelos. Para mitigar esta amenaza, cada participante fue sometido a una entrevista previa al experimento en la cual se constató que disponía del conocimiento suficiente como para participar del mismo.
- **Experiencia previa del modelador en aplicaciones similares**. Al igual que en experimento anterior, haber trabajado en los dominios de las aplicaciones para las cuales se requirió la construcción del modelo de datos puede claramente influir en el tiempo y calidad de los modelos obtenidos por los participantes. Esta amenaza fue mitigada a través de la entrevista previa al experimento en la cual se constató que los participantes no poseyeran ningún tipo de experiencia en los dominios de las aplicaciones descriptas.

7. El Proceso MockupDD general y extensiones

Durante el transcurso de este trabajo de tesis se introdujo la metodología MockupDD mayormente en el contexto del desarrollo Web – lo que se denominó *MockupDD Web*. Esto permitió describir sus procesos y su aplicación en un campo muy popular y conocido en lo concerniente al desarrollo de software actual, el cual también es un área que ha sido abarcada por numerosas metodologías basadas en modelos. En esta sección se describirá el proceso MockupDD en términos generales y luego se mostrarán dos extensiones orientadas al desarrollo de software más específico. De este modo, se brinda un marco y definición generales de MockupDD y se muestra cómo puede aplicarse a través de extensiones o especializaciones para abarcar otras áreas en el desarrollo de software.

7.1. El proceso MockupDD general

En la Sección 3.7 se introdujo el proceso de MockupDD Web. El proceso MockupDD general es un subconjunto del mismo y está conformado por ciertas prácticas, conceptos y herramientas básicas, algunas de las cuales son opcionales según la plataforma y la modelización requerida. Estas prácticas, conceptos y herramientas son:

- User Stories y mockups asociados como artefacto principal y obligatorio de requerimientos.
- El modelo SUI, el cual formaliza la estructura de la interfaz de usuario representada en los mockups.
- El proceso de desarrollo Scrum adaptado para MockupDD.
- La existencia de Procesadores de Mockups (*Mockups Processors* o MP) capaces de transformar rápidamente mockups digitales a modelos SUI.
- La definición de tags y su agrupación en tag sets, donde su sintaxis y contenido puede variar dependiendo de las aplicaciones puntuales de MockupDD. Los tags se caracterizan por tener un nombre, un tag set de pertenencia y un contenido arbitrariamente complejo, como en MockupDD Web.
- La existencia de una Herramienta de Aplicación de Tags (*Tagging Tool* o TT), la cual permita aplicarlos sobre los mockups procesados – y asociarlos al SUI. Esta misma herramienta puede permitir la construcción iterativa del SUI partiendo desde mockups digitales.
- La existencia de uno o más Interpretes de Tags o Generadores de Código (*Tag Interpreters* o *Code Generators* o TI y CG respectivamente) los cuales permitan derivar la aplicación final, ya sea generando su código fuente o bien interpretando los tags.

En el contexto de MockupDD general, el metamodelo SUI y de tags (ver Fig. 6) es exactamente igual al ya presentado. Sin embargo, dependiendo del refinamiento necesario de acuerdo a su área de aplicación, no todos los tags deben ser obligatoriamente asociados a un modelo SUI, el cual en un caso extremo puede omitirse por completo. En este caso, los tags deben ser simplemente dispuestos gráficamente cerca de los elementos del mockup con el solo fin de proveer trazabilidad de requerimientos. En caso de que el uso de

SUI sea bajo o nulo, el uso de las herramientas MP y TT pueden llegar a tener una participación muy baja en el proceso. Por otro lado, al utilizarse mockups de alta fidelidad y dependientes de la plataforma (como HTML en el caso de MockupDD Web) de proveerse una herramienta TT, la misma debe ser re-implementada para proveer las funcionalidades de aplicación de tags y detección de widgets. Una alternativa es la utilización de anotaciones formales en herramientas de mockup digitales como Balsamiq o Pencil para describir los tags. En este último caso, la herramienta MP debe contar con la capacidad de procesar no sólo los mockups sino también sus tags, a fin de generar un modelo SUI taggeado como resultado.

En la Fig. 35 se puede observar un esquema del proceso desde el punto de vista de los artefactos de código. En la figura, más simple que la versión de MockupDD Web (Fig. 4), pueden observarse entre otras cosas la posibilidad de aplicar tags sobre mockups informales en lugar de sobre un modelo SUI y la ausencia de ciertas restricciones asociadas al dominio Web. Es importante destacar que, de no utilizarse un modelo SUI, la metodología especializada de MockupDD tendrá una expresividad reducida – dado que no se dispondrá de toda la semántica provista por la estructura de widgets y sus relaciones al momento de generar código o modelos.

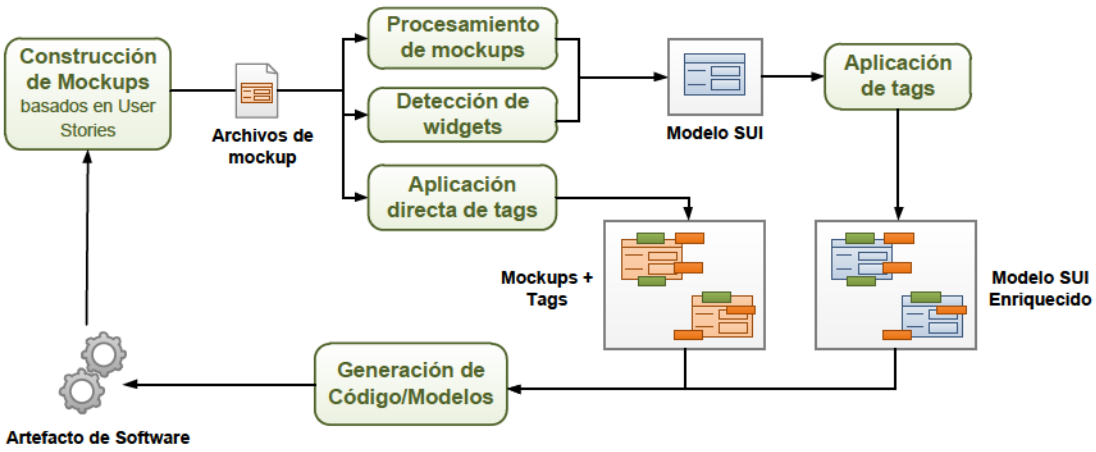


Figura 35. Proceso MockupDD general desde le punto de vista de los artefactos y herramientas utilizadas.

Desde el punto de vista de la implementación, las opciones respecto a cada etapa son muy dependientes de la plataforma, tecnología, dominio y nivel de asistencia particulares relacionados con la especialización de MockupDD a implementar. En lo relativo a la asistencia provista, no es necesario escribir la herramienta MP para cada especialización de MockupDD, sino que puede reutilizarse la misma que ya fue implementado para MockupDD Web, cuya parte del código fuente puede encontrarse online[22]. No obstante, si se desea integrar una nueva herramienta para dibujar mockups al MP existente, deberá escribirse un parser de widgets para la misma (ver Sección 4.4.2). Respecto a la herramienta TT, dada que esta representa la mayor asistencia respecto al tagging, de proveerse debe considerar las siguientes funcionalidades:

- La posibilidad de mostrar interfaces de usuario implementadas en la plataforma actual (sólo su estructura) y enriquecerla de forma no intrusiva para permitir la asociación de tags.

[22] WebSpec Language – https://code.google.com/p/webspec-language/

- Serializadores y deserializadores que permitan convertir los tags de una expresión textual a un objeto de modelo y viceversa. Se recomienda que cada tag sea representado por un tipo concreto de objeto, similar a un Spec Object – se llamarán Tag Objects.
- Si se desea proveer simulación de la aplicación siendo modelada al vuelo o bien su ejecución sin derivación de código desde los tags, entonces debe proveerse un intérprete de Tag Objects que evalúe y aplique los enriquecimientos necesarios a la interfaz de usuario en la plataforma concreta para simular la aplicación.
- Si se desea generar artefactos de software ejecutables, debe proveerse generadores de código capaces de analizar estos Tag Objects y derivar código desde los mismos.

7.2. MockAPI: Focalizando MockupDD al desarrollo de APIs

En esta sección se describirá una especialización de MockupDD llamada MockAPI. La misma permite el modelado y generación de APIs desde mockups taggeados. En las siguientes subsecciones se describirá la motivación detrás de la metodología, su composición, proceso asociado y aspectos de implementación.

7.2.1. Motivación

Como se ha comentado con antelación, las Metodologías Ágiles han mostrado una amplia adopción en la industria [10]. Principalmente, esta adopción se ha debido a que permiten adaptarse rápidamente a requerimientos cambiantes, acortando el ciclo de desarrollo e incluyendo a usuarios finales más activamente en el proceso con el objetivo principal de reducir sus riesgos. Sin embargo, las MA carecen de herramientas y técnicas para capturar requerimientos relacionados con aspectos no visuales, como las APIs que muchas veces forman el *backend* de las aplicaciones y que proveen servicios a otras capas de la misma aplicación o a aplicaciones externas. La causa de esta falencia se debe a que las MA están focalizadas mayormente en requerimientos de interacción (como, por ejemplo, la interfaz de usuario o lógica de negocios) y no tanto en aquellos requerimientos que el usuario no puede observar directamente [42].

En otro contexto, en los últimos años se ha dado un constante crecimiento de las soluciones de Infrastructure-as-a-Service (IaaS) y Software-as-a-Service (SaaS), las cuales están transformando la forma en la que se proveen servicios *en la nube*. Las soluciones IaaS permiten obtener rápidamente recursos de infraestructura (procesamiento, almacenamiento, transferencia de datos, etc.) bajo demanda y a bajo costo. Las soluciones SaaS brindan software bajo demanda en la nube a bajo costo, el cual no necesita desarrollo, deployment ni mantenimiento: el software se abona como un servicio y se utiliza vía Internet. Dado que ambas soluciones proveen sus servicios *en la nube*, requieren de medios para permitir a los usuarios interactuar con soluciones de software on-site para lo cual brindan APIs. Por ejemplo, aplicaciones SaaS como aquellas orientadas a la facturación proveen facilidades para exportar e importar datos hacia y desde sistemas existentes. Las soluciones IaaS permiten, a través de APIs similares, monitorear los recursos de infraestructura utilizados y modificar su uso bajo demanda – por ejemplo, incrementar el almacenamiento virtual o finalizar servidores ociosos.

A nivel de aplicación, las APIs también se han vuelto extremadamente comunes para interactuar con sitios de uso masivo, como las redes sociales Facebook[23] o Twitter[24]. Todas estas aplicaciones en general ofrecen APIs que les permiten a aplicaciones o servicios externos interactuar con su plataforma social – por ejemplo, consultar por gustos personales, contactos, etc. Finalmente, las versiones móviles de estas aplicaciones utilizan las APIs provistas como método de interacción con los servidores.

Por las razones comentadas, el desarrollo de APIs – y más aún, el desarrollo de aplicaciones centradas en APIs – se ha vuelto muy común: en general, una API se considera como un elemento crucial del desarrollo de aplicaciones que corren sobre la nube y requieren soporte para múltiples dispositivos[25]. Sin embargo, debido a las dificultades por parte de las MA a la hora de expresar requerimientos no visibles para el usuario, no siempre es trivial el desarrollo de este tipo de aplicaciones sobre este tipo de procesos.

7.2.2. Propuesta

Para solucionar la problemática planteada, se propone una instanciación de la metodología MockupDD para permitir el desarrollo de aplicaciones *API-First* – es decir, aquellas en las cuales se considera a la API de la aplicación como el artefacto de software más importante a desarrollar. La idea es proveer un proceso y herramientas que permitan la rápida construcción de prototipos de APIs RESTful los cuales puedan ser utilizados para agilizar el desarrollo – por ejemplo, permitiendo testear diferentes frontends, otras APIs o cualquier otro artefacto que haga uso de la API a definir. A su vez, el proceso hace hincapié en utilizar mockups – artefactos de requerimientos que los usuarios finales pueden comprender y utilizar como lenguaje libre de jerga de negocios – a fin de solventar el problema de capturar requerimientos de artefactos que no son directamente visibles para ellos.

Al tratarse de una especialización de MockupDD, el proceso de desarrollo (Fig. 36) en la metodología planteada (la cual se denominó MockAPI [43]) sigue los pasos y reglas del proceso Scrum adaptado para MockupDD (ver Sección 3.7). Sin embargo, en lugar de una Aplicación Web, al final de cada iteración se obtendrá como producto final una API funcional. La API generada representa una versión prototípica pero totalmente funcional de una API REST. Se eligió este tipo de arquitectura por ser uno de los estilos arquitecturales para APIs más usados en la actualidad, presente en APIs emblemáticas como la de Facebook[26].

[23] Facebook Graph API – https://developers.facebook.com/docs/reference/api/
[24] Twitter Developers – https://dev.twitter.com
[25] API-First – http://www.api-first.com/
[26] Facebook Graph API – https://developers.facebook.com/docs/graph-api/using-graph-api/v2.1

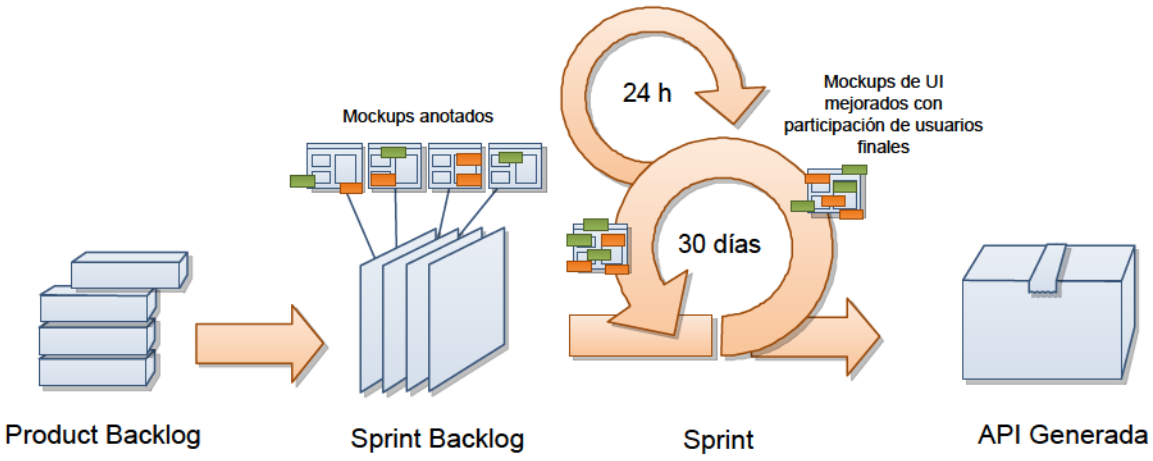


Figura 36. Esquema del proceso de desarrollo planteado por la metodología MockAPI.

En particular, MockAPI no hace uso del SUI, sino que la implementación de su herramienta TT permite aplicar un conjunto simple de tags y a su vez permite relacionarlos entre sí visualmente. Existen en MockAPI dos tipos de tags (`item` y `list`) y respetan la siguiente estructura general:

```
item/list(<resource>)
viewing/editing
[selection]
[paginating]
[sorting]
[ordering]
[navigation(<screen>)]
```

Ambos tipos de tag aceptan los parámetros `viewing` y `editing`, siendo el resto de las especificaciones opcionales y sólo aplicables para el tag `list`. La semántica de cada una de estas especificaciones es la siguiente:

- `list(<resource>)`: De modo similar al tag `Data(*<class>)`, describe una lista de elementos que son mostrados en la interfaz de usuario.
- `item(<resource>)`: De modo similar al tag `Data(<class>)`, expresa que el mockup contiene una representación gráfica de un elemento de tipo `<resource>`.
- `viewing / editing`. Describe las restricciones de acceso al recurso anteriormente introducido por un tag `ítem` o `list`; la primera expresa que el recurso definido por `ítem` o `list` es de sólo lectura, mientras que la segunda define que en la API se podrán realizar operaciones CRUD sobre el mismo.
- `sorting`: Indica que los elementos en la lista (tag `list`) son ordenables por algún criterio.
- `ordering`: Indica que los elementos de la lista pueden ordenarse manualmente por el usuario – por ejemplo, utilizando una UI *drag & drop*.

- `filtering`: Indica que los elementos en la lista pueden ser filtrados de acuerdo a uno o más criterios
- `pagination`: Indica que los elementos pueden agruparse y observarse de a páginas.
- `navigation(<screen>)` indica que el elemento en el mockup permitirá navegar a la pantalla con nombre `<screen>`.

Si bien no se facilita un lenguaje textual para expresar asociaciones entre los tags, las mismas pueden hacerse explícitas a través de un lenguaje gráfico el cual es implementado por la herramienta de soporte TT especializada para MockAPI. En la Fig. 37 pueden observarse ejemplos de tres tags de MockAPI indicando todas sus posibles combinaciones y elementos.

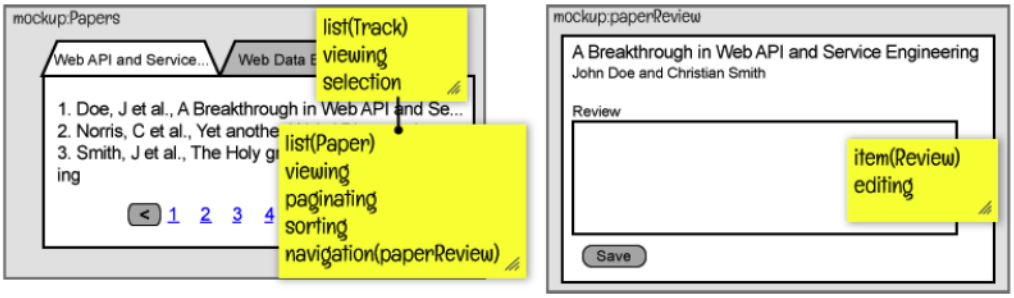


Figura 37. Ejemplo de tres tags MockAPI.

Aquellos aspectos que no puedan expresarse mediante los tags introducidos pueden incluirse en forma de requerimientos textuales utilizando la especificación `specialFeature(<description>)`, donde `<description>` es una descripción en forma de texto para adicionar a los requerimientos más concretos explicitados por las otras construcciones.

El lenguaje de tags de MockAPI (es decir, el metamodelo de los Tag Objects particulares de la metodología) está formalmente graficado en la Fig. 38.

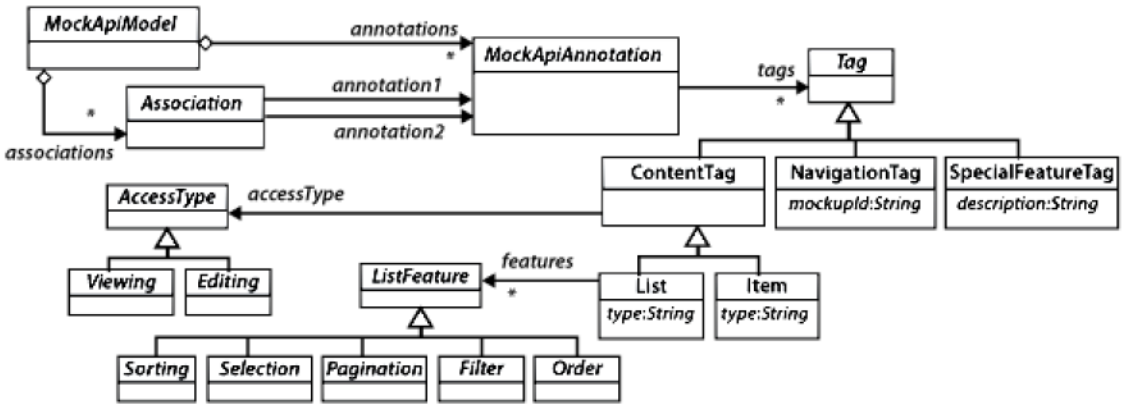


Figura 38. Metamodelo de tags de MockAPI.

Básicamente un modelo MockAPI está compuesto por una serie de anotaciones (`MockApiAnnotation`) y asociaciones entre las mismas (`Associations`). Las `MockApiAnnotations` están conformadas por una serie de tags (clase `Tag`), los cuales representan cada una de las construcciones anteriormente introducidas. Los `Tags` se dividen en `NavigationTags` – representando la construcción `navigation(<screen>)` –, `SpecialFeatureTags` – representando la construcción `specialFeature(<description>)` – y finalmente `ContentTags`. Los `ContentTags` pueden ser de tipo `List` o `Item` y, en el caso de los tags `List`, pueden contener uno o más de los modificadores anteriormente introducidos (representados por las clases `Sorting`, `Selection`, `Pagination`, `Filter` y `Order`. Finalmente un `ContentTag` tiene un modo de acceso, `Viewing` o `Editing`.

7.2.3. Generación de APIs

El proceso de generación de APIs RESTful es similar al proceso de generación de Aplicaciones Web definido con antelación para MockupDD Web. En particular, cada tag es procesado y uno o más artefactos (en este caso, partes de una API RESTful) son generados como producto de este procesamiento. Como ventaja adicional de utilizar este esquema de generación de código y alta abstracción, la API RESTful generada respeta buenos patrones y prácticas definidas [44]. Las reglas de generación definidas son las siguientes:

- `item(resource)` causa la generación de, como mínimo, un endpoint que soporte el método HTTP `GET` sobre `/resources/<id>` si se combina con `viewing`. Combinado con `editing`, se habilitan además los métodos `PUT` Y `DELETE`, los cuales permitirán actualizar y eliminar instancias puntuales de `resource` respectivamente.
- `list(resource)` es similar a `item(resource)`, sólo que además permite manejar listas de elementos, con lo cual inducirá la creación de, como mínimo, un endpoint `/resources` que permita listar todas las instancias de `resource` si se combina con `viewing`. Al combinarse con `editing`, habilita las operaciones `POST` y `DELETE` sobre el endpoint, métodos HTTP los cuales permitirán crear elementos puntuales de tipo `resource` o bien eliminar todos los de su tipo.
- `selection` combinado con `list(resource)`, determina que se permitirá la selección y obtención de instancias individuales de `resource`. Esto implicará la definición de endpoints del estilo `/resources/<id>` aunque no se haya usado `item(resource)`.
- `filter` causará la generación de lógica en la API RESTful para llevar a cabo filtrado de datos. Por ejemplo, asumiendo que se combina con `list(resource)`, `filter` provocará el procesamiento del query string presente en las invocaciones `GET` para filtrar criterios de búsqueda, por ejemplo `/resources?prop1=value1`.
- En el caso de `pagination`, ocurrirá algo similar al caso de `filter` pero orientado a la paginación de datos. Los endpoints que atiendan peticiones `GET` permitirán filtrar por rango de elementos a fin de poder llevar a cabo la paginación especificada, por ejemplo `/resources?from=40&count=20`.

- Las asociaciones entre diferentes tags (graficadas en la Fig. 37) causan la generación de endpoints especiales para relacionar instancias. En el caso del ejemplo de la figura, una instancia de `Track` – `item(Track)` – tiene una lista de `Papers` – `list(Paper)`. Esto provocará no sólo la creación de los endpoints `/papers` y `/tracks`, sino también de `/tracks/<id>/papers`, la cual permitirá acceder a las instancias de `Paper` en el contexto de un `Track` particular (el cual tendrá el id `<id>`). Si la lista permite edición (es decir, incluye el modificador `editing`), entonces la relación será editable – es decir, no sólo se permitirá consultar los `Papers` de un `Track` particular con `GET`, sino también asociarlos entre sí con `PUT`, crear `Papers` nuevos y asociarlos a un track al tiempo con `POST` o eliminarlos de la lista con `DELETE`.

Respecto a las asociaciones, es importante destacar que no sólo las asociaciones gráficas entre tags determinan la generación de recursos asociables. Las relaciones de navegación entre mockups también pueden determinar recursos asociados. Por ejemplo, en el caso de la Fig. 39, se esquematiza dos situaciones diferentes con dos anotaciones similares, en un caso relacionadas explícitamente en un solo mockup y en el otro separadas pero relacionadas entre sí por una navegación. En estos casos en los cuales las relaciones entre anotaciones se hace clara a través de una navegación, las reglas de generación actúan como si las anotaciones estuvieran relacionadas gráficamente entre sí. Es decir, en la Fig. 39, el caso (a) es equivalente al caso (b) en lo que concierne a la API generada.

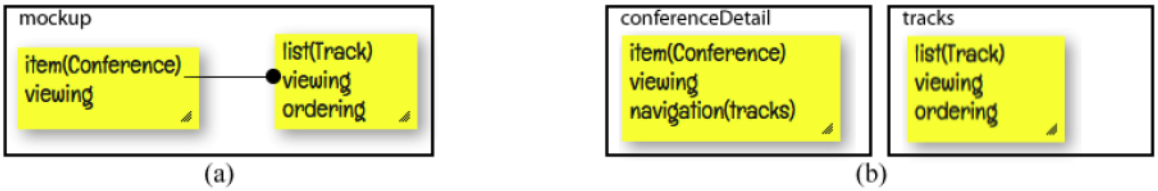


Figura 39. Dos casos equivalentes (en término de la API generada) de asociación entre anotaciones: a través de una asociación explícita (a) o implícitamente a través de una navegación (b).

7.2.4. Implementación

A nivel de implementación, la metología MockAPI está sostenida por una herramienta (la versión personalizada de la herramienta TT, llamada *MockAPI Tool*) la cual permite llevar a cabo la importación y anotación de mockups. Dado que MockAPI no tiene en cuenta el SUI, la herramienta trata los mockups como imágenes y permite aplicalos libremente sobre la misma. Un screenshot de esta herramienta puede observarse en la Fig. 40.

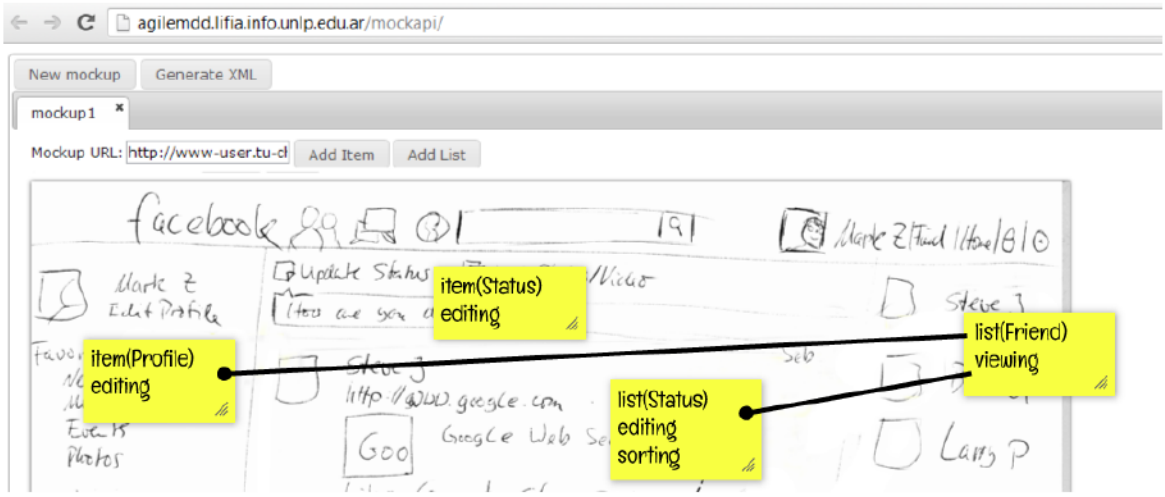


Figura 40. Screenshot de la MockAPI Tool.

La MockAPI Tool facilita las siguientes acciones en el contexto de la meetodología:

1. Importar mockups en forma de imágenes – por lo que no se restringe en absoluto el tipo de mockups.
2. Nombrar y organizar los mockups importados.
3. Aplicar tags y/o asociarlos, validando que su sintaxis sea la correcta.
4. Disparar la inmediata generación de una API RESTful en tiempo real en base a las anotaciones incluidas hasta esta iteración – esta parte de la herramienta actúa como generadora/intérprete de código.

Respecto a la implementación de la API subyacente, en una primer iteración se utilizó el WebComposition/DataGridService (DGS) [45], una Aplicación Web que brinda la posibilidad de crear y configurar servicios web en tiempo real utilizando configuración en XML. La MockAPI Tool procesa las anotaciones y deriva una especificación de la API a generar – la cual consta en una serie de recursos, sus relaciones y sus operaciones permitidas. Esta información estructurada le es provista al DGS, el cual procesa estas especificaciones y configura, bajo demanda y al instante, una API RESTful que implemente lo requerido. Un esquema gráfico de este proceso puede observarse en la Fig. 41. El prototipo generado de la API RESTful es totalmente funcional en el sentido de que puede ser utilizado por cualquier otra aplicación o API. De este modo, basta con importar algunos mockups de la aplicación a desarrollar (por ejemplo, de uno o más frontends gráficos) y anotarlos para obtener una API unificada para agilizar el desarrollo de otros artefactos que la requieran.

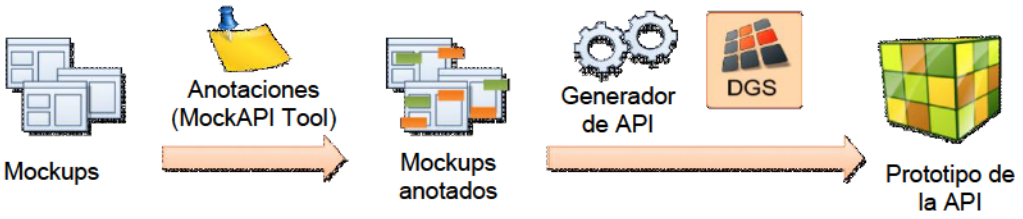


Figura 41. Artefactos involucrados en le proceso de desarrollo de MockAPI.

En la sigueinte sección se abordará un refinamiento de MockAPI el cual permite generar no sólo una API RESTful corriente, sino extenderla en tiempo de ejecución a través de una herramienta web sin romper con las abstracción provista por los tags.

7.3. ELECTRA como extensión de MockAPI

La metodología MockAPI permite generar prototipos de APIs RESTful corrientes en iteraciones cortas y en base a artefactos de requerimientos fáciles de comprender por usuarios finales en pos de agilizar el desarrollo. De este modo, otros artefactos que dependan de la API pueden desarrollarse rápidamente en paralelo utilizándola. Sin embargo, aunque sea reducido, el esfuerzo de diseñar y generar la API desde los mockups es parcialmente desperdiciado debido a que la API generada es sólo un prototipo corriente; esto implica que la misma no puede ser modificada para ser llevada a calidad de producción. Por ejemplo, situaciones muy comunes como validar la estructura de los datos enviados en un request POST de acuerdo a ciertas reglas de negocios o integrar con APIs de terceros son imposibles en el contexto de MockAPI. En este contexto, cualquier extensión que exceda el comportamiento CRUD básico de la API RESTful es imposible de implementar.

Como solución a esta problemática, se replanteó desde cero la implementación de MockAPI y también se rediseñaron los tags para que sean extensibles y permitan la inclusión de funcionalidades más avanzadas y de código interpretado. La metodología implementada se llamó ELECTRA [46] (por *Extensible, modeLdriven Enduser CenTRic API*) y combina la agilidad de la codificación manual y la productividad de la orientación a modelos (basados en tags) de MockAPI para definir APIs RESTful corrientes. En términos de implementación, ELECTRA define un runtime extensible el cual permite la rápida modificación de la API generada intentando preservar la abstracción de los tags siempre que sea posible.

7.3.1. Proceso y tags

Dada su naturaleza extensible, ELECTRA redefine parte del proceso de MockAPI, haciéndolo más detallado (ver Fig. 42).

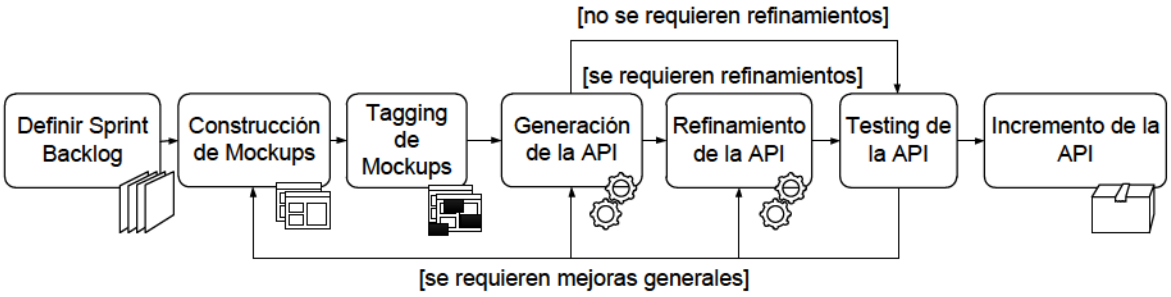


Figura 42. Proceso de la metodología ELECTRA.

A diferencia de MockAPI, luego de generar la API desde los mockups anotados ELECTRA permite, de ser necesario, una etapa de refinamiento en la cual pueden modificarse la implementación generada utilizando codificación manual e intentando preservar al máximo posible la abstracción de cada tag. Sin embargo, el proceso sigue estando centrado en el tagging de mockups. Aunque el runtime que corre la API RESTful fue reescrito en su totalidad, la herramienta TT de MockAPI fue reusada en su totalidad como base para la implementación de la Tagging Tool de ELECTRA.

Respecto a los tags, ELECTRA los divide en tres tag sets particulares, cada uno con una sintaxis concreta:

- *Tags de Datos*, los cuales especifican los diferentes tipos de objetos que la API manejará, basándose en entidades de negocios planteadas por los clientes o usuarios finales
- *Tags de Restricciones (Constraints)*, los cuales permiten especificar reglas de negocios que la API debe cumplir. Son especificados en lenguaje textual y comprensible para los usuarios finales, pero pueden escribirse también en código ejecutable.
- *Tags de Acciones (Action)*, los cuales describen la ejecución de tareas complejas o heterogéneas como consecuencia de la manipulación de datos por parte de la API

Los tags de Datos, los cuales son muy similares a los de MockAPI, son el núcleo de ELECTRA dado que permiten definir la estructura general de la API sobre la cual luego de aplicarán los tags de Restricciones y Acciones. Los mismos consisten en una simplificación de los tags de MockAPI, reduciéndolos a `List/Item` y `Viewing/Editing` (Fig. 43), permitiéndose también su asociación.

```
Item(type)                  List(type)
(Viewing|Editing)           (Viewing|Editing)
```

Figura 43. Estructura de los tags de datos de ELECTRA.

Los tags de Restricciones se aplican asociados a los de Datos y actualmente pueden ser de dos tipos: `Attributes` o `Validate`. El tag `Attributes` permite describir las propiedades que va a poseer un tipo de objetos de negocios particular definido por el tag de Datos asociados. Estos tags permiten validar los requests enviados a la API contra un esquema de propiedades determinado, controlando que se envíen los atributos correctos con sus tipos de datos y sintaxis adecuados.

El tag `Validate` permite especificar reglas generales de validación sobre los objetos de negocios definidos en los tags de Datos. Este tag admite dos modos: textual y ejecutable – una estrategia similar a la

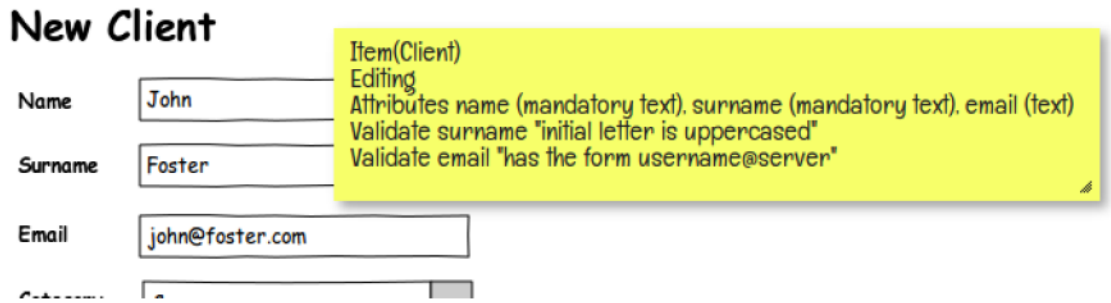
ya introducida para MockRE. El primer modo está orientado a clientes o usuarios finales y permite expresar requerimientos de validación de un modo textual que ellos pueden comprender, los cuales a su vez quedan registrados para los desarrolladores. Luego, el equipo de desarrollo puede transformar esa descripción textual en código ejecutable el cual implemente el requerimiento de validación deseado en términos concretos.

En la Fig. 44 se muestra la sintaxis formal de un tag de Validación junto con dos ejemplos de su aplicación con sus dos modos soportados en el contexto de una aplicación típica de facturación. En la figura se muestra puntualmente un mockup que permite la carga de un nuevo cliente. Es importante destacar que, de modo similar a MockupDD Web, los diferentes tipos de tags pueden introducirse iterativamente durante el desarrollo a medida que se van requiriendo; por ejemplo, primero puede obtenerse una API funcional sin tags de Validación para disponer rápidamente de un prototipo de la API y luego en iteraciones posteriores agregar las reglas necesarias a través de los mismos.

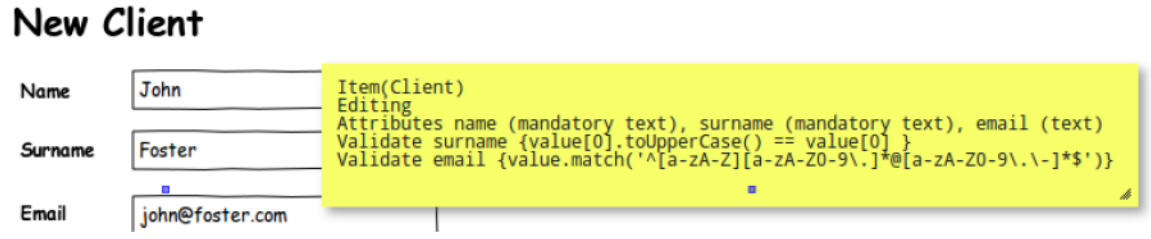
```
Attributes <attribute> [, <attribute>]*
<attribute> = <attributeName> ([mandatory] <type>)
<type> = text | number | decimal | date | logic | content
<attributeName> = identifier

Validate <attributeName> <textOrCode> [; <errorCode> ; <errorDescription>]
<attributeName> = identifier
<errorCode> = identifier
<textOrCode> = "string" | {string}
```

(a) Gramática de los tags de Validación



(b) Tags de Validación en modo textual



(c) Tags de Validación en modo ejecutable

Figura 44. Gramática y ejemplos de tags de validación en sus diferentes modos

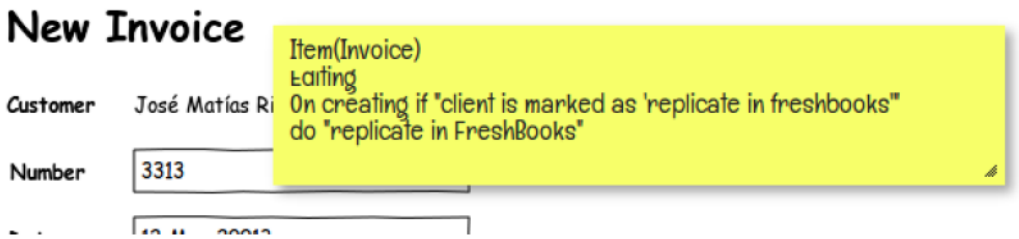
Ahondando un poco más en las características de los tags de validación, `Attributes` permite definir los nombres de los atributos que tendrán los objetos, su carácter de obligatorio u opcional y su tipo de datos. En

el caso del ejemplo de la Fig. 44, se describe la entidad `Client` la cual tendrá los atributos `name`, `surname` e `email`, los 3 textuales y los dos primeros obligatorios (`mandatory`). Las reglas de validación introducidas por `Validate` primero se escriben textualmente (Fig.44.b) y luego se refinan en código (Fig.44.c). Un tag `Validate` hace referencia a un atributo particular y permite escribir una regla de validación para el mismo. Los tags almacenan ambas versiones de su especificación (textual y código) lo cual hace posible ir de una a otra de ser necesario (por ejemplo, para revisar las especificaciones o modificarlas e interactuar con usuarios finales). El lenguaje de programación utilizado en ELECTRA es JavaScript, dado que por sus características dinámicas y por su variedad de intérpretes bajo diferentes tecnologías y plataformas resulta flexible para escribir fragmentos de código a ser interpretado en los tags.

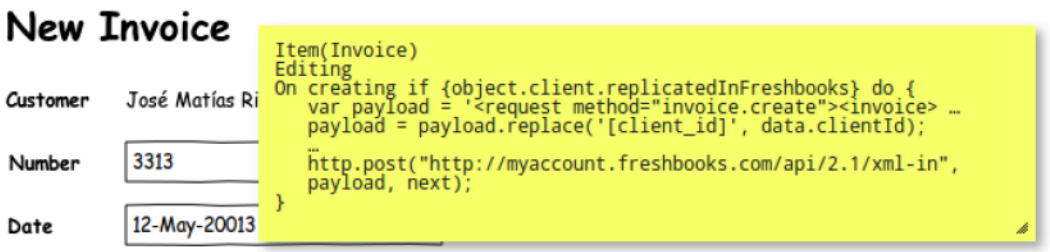
Respecto a los tags de Acciones, al igual que el tag Validate, los mismos tienen un modo textual y un modo ejecutable. En la Fig. 45 se muestra su gramática y ambos modos.

```
On <method> [if <condition>] do <textOrCode>
<method> = create | update | delete | get
<condition> = <textOrCode>
<textOrCode> = "string" | {string}
```

(a) Gramática de un tag de Acción



(b) Tag de Acción en modo textual



(c) Tag de Acción en modo ejecutable

Figura 45. Gramática y ejemplo de un tag de Acción.

En concreto un tag de Acción se ejecuta como efecto colateral de llevar a cabo una operación de la API RESTful – por ejemplo, la creación o eliminación de un objeto. En su forma textual, el tag especifica bajo qué evento se ejecutará (creación, actualización, borrado u obtención, representadas por las palabras clave `create`, `update`, `delete` y `get` respectivamente) y una descripción textual de la acción que se llevará a cabo. Opcionalmente, puede proveerse también en texto plano una condición bajo la cual se ejecutará la

acción en el contexto de la operación particular. En su modo ejecutable, ambas especificaciones textuales (condición y acción) se refinan con el código concreto a ejecutar.

En la Fig. 45, se describe la acción a llevar a cabo luego de la creación de un objeto de tipo `Invoice` – una factura, en el contexto de la aplicación de facturación del ejemplo anterior. En la Fig. 45, la acción a ejecutar integrará con la API de Freshbooks[27] (una reconocida aplicación de facturación online) para replicar la factura siempre y cuando se haya habilitado esta opción de replicación para el cliente asociado.

Además de ejecutar el código interpretado en JavaScript, ELECTRA brinda una serie de objetos *helper* para los desarrolladores los cuales permiten ejecutar acciones habituales a la hora de implementar APIs. Un ejemplo de este tipo de objetos es `http`, el cual permite ejecutar peticiones HTTP para integrar con otras interfaces.

7.3.2. Arquitectura e implementación

Con el fin de hacer corresponder y aislar cada anotación a un artefacto concreto de implementación a fin de preservar la abstracción al máximo posible, la API generada por ELECTRA se estructuró en base al patrón de integración *Pipes and Filters* [47]. Este patrón fue elegido debido a su simplicidad, la cual lo ha llevado a aplicarse en la mayoría de los frameworks MVC Web. El patrón se compone por una serie de *Filtros* interconectados en serie, en donde cada uno toma una entrada y genera una salida; la salida del n-ésimo Filter es la entrada del (n + 1)-ésimo en la cadena – llamada *Pipe*. En el caso de ELECTRA, cada tag genera un Filter particular (implementado como un script JavaScript interpretado en el server-side) el cual implementa una parte de la semántica necesaria de la API. Esto permite que, de ser necesario, se pueda refinar el código fuente generado por cada tag particular sin necesidad de romper la abstracción en todos los otros casos. La Fig. 46. La ejecución del primer paso en el Pipe (`filterList`) consiste en un script que devuelve la lista de Filters a aplicar sucesivamente. De ser necesario agregar Filters intermedios codificados manualmente a los ya derivados por los tags, los mismos pueden referenciarse a esta lista con el fin de que sean invocados cuando correspondan durante la ejecución del Pipe.

Para ejemplificar la extensibilidad general de ELECTRA, se mostrará cómo se pueden implementar 3 US adicionales para el ejemplo de la aplicación de facturación introducida hasta el momento. Las US serán las siguientes:

- **US1**. Como usuario, deseo que mis facturas incluyan productos que yo he registrado con antelación en lugar de sólo texto.
- **US2**. Como usuario, deseo no permitir la creación de facturas menores a 100 pesos – debido a políticas internas de la empresa.
- **US3**. Como un usuario, deseo que, en el caso de ciertos clientes particulares, mis facturas sean subidas a Freshbooks.

Dos de estos tres casos de uso pueden ser implementados refinando anotaciones y uno tocando sólo un artefacto de código. En la Fig. 47 se muestra qué artefactos concretos (scripts o tags) hay que modificar o

[27] Freshbooks – http://freshbooks.com

agregar en cada caso, dejando el resto de los artefactos intactos. La figura además muestra que, al no haber *colisiones* entre los diferentes artefactos impactados (al menos en el caso del ejemplo), las 3 US pueden ser atacadas por diferentes desarrolladores en paralelo.

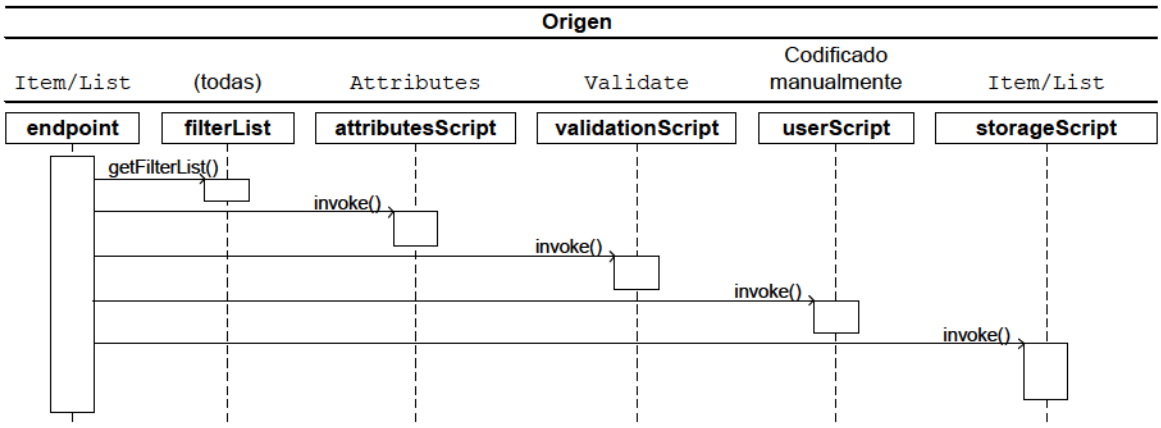


Figura 46. Ciclo de vida de un request, arquitecturado con ELECTRA

	endpoint	filterList	attributesScripts	validationScript	userScript	storageScript	freshbooksScript
US1						Almacenar y recuperar producto desde código	(no definida hasta este momento)
US2				Agregar 2 reglas de validación			(no definida hasta este momento)
US3		Agregar script al final del pipe					Implementar invocación a API de Freshbooks

Se tiene que modificar manualmente un script, rompiendo la abstracción del tag
Se tiene que modificar manualmente un script, sin romper la abstracción de ningún tag
Se modifica una anotación – y, por lo tanto, se regenera su script, preservando la abstracción

Figura 47. Artefactos a modificar para implementar la funcionalidad extra requerida.

7.3.3. Conclusión

La metodología ELECTRA plantea una extensión a MockAPI en la cual se brinda dualidad de tags (textuales y para desarrolladores), siguiendo la estrategia de MockRE. En adición, ELECTRA brinda también un esquema de generación de código que permite su sencilla extensión a través de codificación manual. Tanto MockAPI como ELECTRA están basadas en MockupDD y la clave de sus procesos de desarrollo es el uso y anotación de mockups. Sin embargo, mientras que MockAPI permite generar APIs RESTful prototípicas para agilizar el desarrollo, ELECTRA extiende esta idea proponiendo extensibilidad y refinamientos con el objetivo de reusar esta API y obtener, luego de los refinamientos pertinentes, una API con calidad de producción.

Todos los aspectos de las APIs generadas por ELECTRA pueden refinarse: aspectos de validación, integración, almacenamiento de datos o cualquier otro que se requiera, incluyendo scripts de usuario que no rompen la abstracción de lo ya implementado. De este modo, ELECTRA plantea no sólo un esquema de modelado ágil y extensible utilizando mockups (como propone MockupDD, metodología en la cual fue basada), sino también extensible por codificación manual, reduciendo los impactos de la misma en los modelos (planteados por el uso de tags).

7.4. Conclusión general

En esta sección se describieron las características generales del proceso MockupDD, abstrayéndolo de su especialización web (MockupDD Web) la cual fue descripta al principio de este trabajo de tesis. El proceso MockupDD *general* sigue las mismas guías generales introducidas en MockupDD Web, sólo que desestructura algunas partes del proceso y flexibiliza la necesidad de algunos artefactos – como por ejemplo, el modelo SUI. Esto brinda mayor libertad para especializar e instanciar la metodología a áreas o usos particulares y para adaptarla a conveniencia de los desarrolladores y modeladores.

Una vez introducida la metodología general, se describió MockAPI, una instanciación de MockupDD la cual utiliza mockups y tags para definir APIs RESTful con el objeto de proveer mayor productividad en el desarrollo de aplicaciones API-First. Esta metodología fue luego refinada en ELECTRA, también descripta en esta sección, en la cual se muestra cómo se pueden combinar la generación de código interpretado a partir de tags y su refinamiento a través de codificación manual.

8. Trabajos y propuestas relacionadas

En esta sección se describirán diferentes trabajos relacionados a MockupDD desde diferentes perspectivas: formalización de requerimientos de interacción, uso de especificaciones de UI en el desarrollo, uso de anotaciones para enriquecer artefactos de requerimientos, entre otros. Cada perspectiva será abordada en una subsección particular.

8.1.1. Especificación de requerimientos de interacción

En el tópico de MDD, se ha documentado el uso de prototipos o modelos estructurales de interfaces de usuario entre diferentes enfoques MDWE. En el contexto del OO-Method, Panach et al. proponen obtener requerimientos de interacción desde bocetos de interfaz de usuario [48], creando modelos de tareas estructurales desde los mismos utilizando el formalismo de ConcurTaskTrees [49]. También se han registrado avances en la integración de especificaciones de UI con lenguajes de requerimientos, como el caso de RSL [15]. En este último, un modelo general de interfaz de usuario es utilizado para especificar el tipo de objetos de dominio que la UI manejará y cómo los mismos son manipulados en un metamodelo de *storyboard* no lineal. Este tipo de metamodelo expresa el tipo de interacción levada a cabo en la UI pero sin indicar el orden en el que los pasos que componen esta interacción son llevados a cabo. En lugar de focalizarse en la generación de modelos de tareas como en [48], MockupDD Web en particular se focaliza en generar modelos de aplicaciones data-intensive para metodologías MDWE ya existentes como WebML. En general, la metodología MockupDD propone la generación de software funcional como última etapa de su proceso de desarrollo iterativo, dándole extrema importancia a que el código o modelo generado sea testeable por usuarios finales.

Varias estrategias y metodologías han sido propuestas para modelar requerimientos de interacción – con, en algunos casos, la posibilidad de generar código corriente de desde estas especificaciones. El ya mencionado lenguaje RSL es uno de ellos, en el cual se utilizan storyboards semi-formales para modelar cómo se comportará la aplicación utilizando mockups. WebSpec [50], un lenguaje de dominio específico visual diseñado para capturar requerimientos de interacción, utiliza descripciones formales como precondiciones y aserciones lógicas para especificar cómo deberá actuar la interfaz de usuario en cada User Story elicitada. Este lenguaje está focalizado en detallar requerimientos de interacción (es decir, cómo la aplicación se tiene que comportar frente a estímulos del usuario) en lugar de definir sus características subyacentes como objetos de su dominio, sus relaciones y cómo los mismos deben ser manipulados. En este contexto, MockupDD en general está orientado y restringido a estas últimas características en pos de generar aplicaciones o artefactos de software corrientes (como por ejemplo, una API) que puedan ser probadas por usuario finales o clientes o integradas con otros artefactos de software lo más rápido posible. Otra propuesta similar en este contexto es MoLIC [51], un lenguaje de modelado para diseñar interacción como metáfora de una conversación entre el usuario y diseñadores. La base de este lenguaje son los Diagramas de Interacción MoLIC, los cuales muestran la interacción de a turnos entre diseñador y usuario, formando hilos de conversación que se estructuran en tópicos y sub-tópicos. MoLIC está diseñado para resolver al máximo

posible las ambigüedades y errores de interpretación en las especificaciones de requerimientos. MockupDD intenta resolver el mismo problema utilizando descripciones cercanas y, de ser posible, totalmente comprensibles por usuarios finales – como se ha comentado en el caso de MockRE y ELECTRA.

8.1.2. Uso de especificaciones de interfaces de usuario en los procesos de desarrollo

Diferentes y variados tipos de especificaciones de interfaz de usuario han sido utilizadas tanto en el contexto de enfoques Model-Driven así como también en metodologías ágiles. Por un lado, se ha documentado el exitoso uso de mockups en procesos ágiles. En [13], se resumen diferentes experiencias del uso de mockups de UI en este contexto. Resultados estadísticos descriptos en el este trabajo han mostrado que los mockups resultan esenciales para asegurar características fundacionales de usabilidad y también representan un agregado muy valioso a las User Stories para describir requerimientos. En este contexto, también se han comentado sus ventajas para estimar los efectos concretos del desarrollo de cada User Story [14]. En [12] se han alcanzado conclusiones similares, comentado que los desarrolladores admiten darle más importancia a los mockups que a las User Stories asociadas a los mismos. En este trabajo se ha mostrado además cómo pueden utilizarse mockups digitales durante todo el proceso de desarrollo a utilizando una metodología de desarrollo MDWE centrada en el usuario.

Otra práctica común relativa al uso de mockups es asociarlos a artefactos de requerimientos y diseño tradicionales. En [52] los autores describen la combinación de mockups con Use Cases UML con el objeto de derivar prototipos de la aplicación siendo desarrollada. Esta combinación también se ha mostrado en el trabajo de Kulak et al. [53]. También se ha reportado la combinación de mockups y User Stories en el contexto de desarrollos ágiles [16]. Anotar mockups informalmente se ha registrado también como una práctica frecuente en la etapa de captura de requerimientos [20,54]. En el trabajo de Ramdoyal et al. [55] se ha comentado el uso de mockups estructurados en forma de formularios para descubrir requerimientos de contenido y generar una parte de la aplicación. No obstante, en este trabajo se limita al uso de mockups estructurados como formularios, lo cual no siempre puede aplicarse de acuerdo a la aplicación que se desarrolle. Una idea similar se describe en el proceso ICONIX [56], el cual propone comenzar el desarrollo con prototipos de interfaces de usuario como artefactos de requerimientos principales. En esta metodología, los mockups son utilizados como estrategia de obtención de requerimientos de comportamiento y, al mismo tiempo, sirven para construir una primera versión del modelo de dominio. MockupDD propone generar las aplicaciones o artefactos de software completos a través del uso de anotaciones sobre mockups, las cuales además son técnicamente valorables para los desarrolladores y, al mismo tiempo, también simples de comprender para los usuarios finales.

8.1.3. Enriquecimiento de artefactos de requerimientos a través de anotaciones

La idea de anotar artefactos de requerimientos conocidos no es necesariamente nueva. En [57], se propone anotar Casos de Uso [53] con especificaciones de modelado de datos – algo similar a los tags `Data()` que propone MockupDD Web. El fin de esta propuesta es la especificación y construcción iterativa de modelos de datos utilizando artefactos de requerimientos comunes y aceptados por la industria y la

academia. El prototipado de mockups utilizando anotaciones también ha sido propuesta por Constantine et al. a través de sus llamados Canonical Abstract Prototypes (CAPs) [58]. Bajo esta estrategia, se utilizan mockups de baja-media calidad en donde cada elemento (llamado Componente) tiene un estereotipo particular. Los Componentes generales están divididos en dos clases principales: materiales y herramientas. Los primeros representan a los contenedores, contenido, información, datos o cualquier otro objeto de interfaz de usuario a manipular u operar en la ejecución de alguna tarea. Las herramientas, por otro lado, permiten materializar acciones, operaciones, mecanismos o controles que son utilizados para crear, manipular, transformar u operar sobre materiales. Los diferentes tipos de componentes de los Canonical Abstract Prototypes han sido refinados para expresar, por ejemplo, operaciones de selección y creado de objetos (herramientas) y listas de elementos y elementos editables (materiales). Si bien puede hacerse una analogía entre los Componentes de los CAPs y los elementos taggeados del SUI de MockupDD y, además, ambas metodologías utilizan prototipos de interfaces de usuario como base para especificar requerimientos, existen varias diferencias entre ambas. En primer lugar, los autores de los CAPs proponen enlazar estos prototipos con interfaces de usuario más detalladas. Esto seguiría el ciclo Model-Driven clásico que MockupDD intenta romper, modelando sobre mockups detallados en lugar de partir de modelos abstractos de interfaz de usuario y refinarlos. Esto permite obtener artefactos ejecutables y presentación similar a la aplicación final desde el principio del desarrollo en lugar de posponer su ejecución hasta etapas más avanzadas – a fin de reducir los riesgos de posibles malas interpretaciones de requerimientos, entre otras cosas. Por otro lado, los CAPs no proponen un modo formal de relacionar los diferentes componentes utilizados en los prototipos y, por lo tanto, su capacidad para expresar patrones de interfaces de usuario más complejos de un modo concreto (por ejemplo, Two-Panel Selector [59]) es más limitado – en general se hace utilizando anotaciones textuales informales. Finalmente, los CAPs no proponen ninguna estrategia para expresar formalmente o de un modo ejecutable aquellos requerimientos que no pueden expresarse con su lenguaje de Componentes. MockupDD y, en particular, MockupDD Web intentan solventar estas cuestiones proveyendo formas de relacionar los tags y los widgets SUI (por ejemplo, utilizando la sintaxis `<widget-id>:<typeRef>`) y proveyendo tags generales como `Query()` o `Action()` los cuales permiten introducir código ejecutable como especificación. Como se ha demostrado anteriormente, estos refinamientos orientados a desarrolladores pueden aplicarse en conjunto con visiones orientadas a usuarios finales utilizando un metamodelo común subyacente y más de una sintaxis textual.

8.1.4. Otros trabajos relacionados

Respecto a las implementaciones planteadas para MockupDD y, en particular, en el contexto de MockAPI y ELECTRA, existen varias propuestas dirigidas por modelos para definir APIs RESTful. Entre ellas, se ha mostrado cómo puede extenderse el método OOH4RIA para soportar operaciones RESTful proveyendo un proxy a las operaciones a ejecutar sobre el modelo de datos subyacente [60]. En [61] se describe una estrategia similar, sólo que utilizando estereotipos sobre modelos UML. Finalmente, la

metodología WebML también brinda soporte para la definición de APIs RESTful[28]. Sin embargo, ningunas de estas estrategias tiene como objetivo resolver la problemática de proveer una forma clara de definir y extender APIs RESTful utilizando codificación directa como lo hace ELECTRA. A su vez, estos métodos también dejan de lado el uso de artefactos de requerimientos que sean fácilmente comprensibles por usuarios finales y desarrolladores.

El autor de este trabajo ha reunido también una serie de publicaciones las cuales muestran los orígenes, evolución y también ideas colaterales de MockupDD. En principio, se describió el uso de mockups como un artefacto inicial para generar interfaces de usuarios basadas en modelos [36]. Este trabajo ha sentado las bases teóricas y prácticas del Structural Interface Model, el modelo fundamental sobre el cual se basan el resto de las especificaciones (tags) en MockupDD. En [33] y [34], se describe la generación de modelos UWE desde mockups anotados, utilizando el SUI como base. En [6] se describe una estrategia similar en el contexto de WebML. Ambos trabajos han sido combinados para lograr un lenguaje de tags unificado, general para metodologías MDWE el cual sienta las bases de MockupDD Web.

[28] WebRatio – Getting Started with Web Services – http://www.webratio.com/learn/article/Getting-started-with-Web-Services

9. Conclusiones y Trabajo Futuro

En este trabajo se presentó MockupDD, una propuesta de desarrollo de software dirigida por modelos y fuertemente basada en mockups de interfaz de usuario. Su objetivo primordial es combinar las características positivas de las metodologías MDD y Ágiles. Durante la primera parte de esta tesis y con el propósito de introducir MockupDD en un campo particular, conocido y explotado, se introdujo MockupDD Web, una especialización de MockupDD orientada a la definición de Aplicaciones Web data-intensive. MockupDD Web combina las prácticas tradicionales de MDWE con algunas características de procesos de desarrollo Ágiles. Dado que el área de MDWE ha sido extensamente estudiada en las últimas dos décadas, el enfoque MockupDD Web no ha propuesto una alternativa de modelado más, sino que se ha definido como un *agilizador* de propuestas MDWE existentes y también como una técnica de captura de requerimientos orientadas a la construcción de Aplicaciones Web data-intensive.

Se ha incluido como parte del trabajo la descripción de dos evaluaciones llevadas a cabo a través de experimentos controlados las cuales han demostrado que MockupDD Web es capaz de mejorar el proceso de construcción de modelos, haciéndolo más eficiente y menos propenso a errores. Adicionalmente, se ha observado empíricamente a través de la evaluación llevada a cabo que MockupDD permite introducir características ágiles al proceso de modelado gracias a su técnica de modelado basada en mockups. Más precisamente:

- A través del uso de mockups, se mejora la captura de requerimientos en comparación a métodos textuales [8] y, a su vez, también facilitan la comunicación entre usuario finales y desarrolladores.
- La estrategia de modelado atómico e iterativo facilitada por los tags y la rápida generación de aplicaciones corrientes permiten a los usuarios finales probar la aplicación siendo construida y proporcionar devoluciones durante las iteraciones
- Utilizando la totalidad de las herramientas y técnicas propuestas por MockupDD, se mejora la trazabilidad de requerimientos a través de (1) la utilización tags en modo textual que los usuarios y clientes pueden comprender y (2) su asociación obligatoria a mockups los cuales funcionan como un lenguaje compartido entre stakeholders y desarrolladores. Esto también fue comprobado estadísticamente en las evaluaciones.

Luego de haber introducido MockupDD en términos concretos a través de su especialización Web, se describió MockupDD de un modo abstracto (lo que se llamó *MockupDD general*). Este último detalló las características fundacionales de la metodología y flexibilizó algunos de los aspectos de MockupDD Web con el objeto de hacer que la misma sea más adaptable a ciertos usos y tópicos puntuales. Finalmente, se mostró cómo MockupDD general puede especializarse para un dominio particular – la definición de APIs RESTful. Esto dio origen a las metodologías MockAPI y su refinamiento, ELECTRA.

Respecto al trabajo actual y futuro, como objetivo a corto plazo en el contexto de MockupDD Web se espera realizar validaciones adicionales a la incluida en este trabajo, incluyendo un abanico más amplio de

aplicaciones y desarrolladores. Se espera que los resultados de estas validaciones ayuden a mejorar la herramienta y la metodología en general. Adicionalmente, se están analizando diferentes *concerns* a considerar en el lenguaje a través de la inclusión de nuevos tag sets. Algunos de estos concerns son comportamientos RIA, características sociales y la especificación de restricciones de seguridad en las aplicaciones. En este trabajo se describió la integración de MockupDD Web con UWE y WebML y se hizo hincapié en esta última metodología gracias a la cantidad de herramientas y soporte disponibles para la misma. Sin embargo, se considera altamente importante el soporte de otras metodologías como OOHDM, a fin de poder asegurar la aplicabilidad de MockupDD Web en el campo MDWE en general. Adicionalmente, se está construyendo un catálogo de patrones de interfaces de usuario utilizados frecuentemente, implementados con modelos SUI y tags a fin de corroborar la implementación de configuraciones tradicionales de interfaces de usuario con la metodología.

Otro campo interesante como trabajo futuro consiste en unificar parte de las implementaciones de ELECTRA y MockupDD Web. En particular, el runtime de ELECTRA plantea una arquitectura extensible para generar APIs RESTful en tiempo de ejecución. Esto permitiría utilizar su infraestructura para implementar un Demo Sandbox capaz no sólo de generar una versión de demostración client-side, sino también su parte server-side, acercando la calidad de las aplicaciones demostradas a una aplicación de producción.

La metodología MockupDD, como fue descripta en este trabajo de tesis, fue planteada en un contexto de desarrollo completo de una aplicación desde sus comienzos. No obstante, dado que la metodología basa fuertemente en el uso de especificaciones de UI Mid-Fi y Hi-Fi (desde mockups construidos con herramientas como Balsamiq hasta HTML), también puede aplicarse como un método de extensión o aumentación de aplicaciones existentes. En este contexto, se está trabajando en la posibilidad de especializar MockupDD para soportar la especificación y modelado de requerimientos de aumentación, en particular en el área Web [62] – lo que se conoce como *Web Augmentation*.

Fuera del campo MDWE, MockupDD puede ser utilizado para especificar artefactos no necesariamente relacionados con elementos de interfaz de usuario. Otra línea de trabajo interesante se relaciona con el uso de metáforas visuales que facilitan los mockups para especificar aspectos como procesos de negocios, transacciones, concurrencia, definiciones de modelos de tareas, etc. Al utilizar mockups anotados en este contexto, los tags pueden ayudar a descubrir requerimientos funcionales o no funcionales en las aplicaciones los cuales no son necesariamente visibles a clientes o usuarios finales.

10. Abreviaturas

En esta sección se describe el significado de las abreviaturas generales utilizadas en la presente tesis.

Acrónimo	Descripción
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BFS	Breadth-First Search
CRUD	Create, Read, Update and Delete
DOM	Document Object Model
GQM	Goal-Question-Metric
HQL	Hibernate Query Language
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IFML	Interaction Flow Modeling Language
JSON	JavaScript Object Notation
MA	Metodologías Ágiles
MDD	Model-Driven Development
MDWE	Model-Driven Web Engineering
MVC	Model-View Controller
OO	Orientación a Objetos – Object Orientation
OOHDM	Object-Oriented Hypermedia Design Method
REST	Representational State Transfer
RIA	Rich Internet Application
UCD	User-Centered Design
UI	User Interface
UML	Unified Modeling Language
US	User Story
UWE	UML-based Web Engineering
WebML	Web Modeling Language
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XPath	XML Path Language
XSD	XML Schema Definition
XUL	XML User Interface Language

11. Publicaciones realizadas durante el Doctorado

Todo el contenido descripto en esta Tesis ha sido publicado oportunamente publicado en workshops, congresos y revistas, todos ellos con referato. A continuación, se detalla la lista de publicaciones completas, cuyo contenido ha sido explayado en este trabajo de tesis.

- J. M. Rivero, G. Rossi, J. Grigera, J. Burella, E. Robles Luna, and S. Gordillo, "From Mockups to User Interface Models: an Extensible Model Driven Approach," in Proceedings of the 10th International Conference in Web Engineering (ICWE'10), 2010, pp. 13–24.
- J. M. Rivero, G. Rossi, J. Grigera, E. R. Luna, and A. Navarro, "From Interface Mockups to Web Application Models," in 12th International Conference on Web Information System Engineering, 2011, pp. 257–264.
- J. M. Rivero, J. Grigera, G. Rossi, E. Robles Luna, and N. Koch, "Improving Agility in Model-Driven Web Engineering," in Proceedings of the 23rd International Conference on Advanced Information Systems Engineering, 2011, pp. 163–170.
- J. Grigera, J. M. Rivero, E. R. Luna, F. Giacosa, and G. Rossi, "From Requirements to Web Applications in an Agile Model-Driven Approach," in Proceedings of the 12th International Conference in Web Engineering, 2012, vol. 7387, pp. 200–214.
- J. M. Rivero, J. Grigera, G. Rossi, E. R. Luna, and N. Koch, "Towards Agile Model-Driven Web Engineering," Lect. Notes Bus. Inf. Process., vol. 107, pp. 142–155, 2012.
- J. M. Rivero, S. Heil, J. Grigera, M. Gaedke, and G. Rossi, "MockAPI: An Agile Approach Supporting API-first Web Application Development," in Proceedings of the 13th International Conference on Web Engineering, 2013, pp. 7–21.
- J. M. Rivero and G. Rossi, "MockupDD: Facilitating Agile Support for Model-Driven Web Engineering," in ICWE 2013 International Workshops, 2013, vol. 8295, pp. 325–329.
- J. M. Rivero, E. Robles Luna, J. Grigera, and G. Rossi, "Improving user involvement through a Model-Driven requirements approach," in 2013 International Workshop on Model-Driven Requirements Engineering (MoDRE),, 2013, pp. 20–29.
- J. M. Rivero, J. Grigera, G. Rossi, E. R. Luna, F. Montero, and M. Gaedke, "Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering," Inf. Softw. Technol., pp. 1–18, Jan. 2014.
- D. Firmenich, S. Firmenich, J. M. Rivero, and L. Antonelli, "A Platform for Web Augmentation Requirements Specification," in Proceedings of the 14th International Conference in Web Engineering. In Press, 2014.
- E. Robles Luna, J. M. Rivero, M. M. Urbieta, and J. Cabot, "Improving the scalability of Model driven Web engineering approaches with runtime transformations," in Proceedings of the 14th International Conference in Web Engineering. In Press, 2014.

- E. Robles Luna, J. M. Rivero, and M. M. Urbieta, “Abstractions: A model based environment for developing high scalable Web applications,” in Proceedings of the 14th International Conference in Web Engineering. In Press, 2014.
- J. M. Rivero, S. Heil, J. Grigera, E. Robles Luna, and M. Gaedke, “An extensible, Model-Driven and end-user centric approach for API building,” in Proceedings of the 14th International Conference in Web Engineering. In Press, 2014.
- J. Grigera, A. Garrido, and J. M. Rivero, “A Tool for Detecting Bad Usability Smells in an Automatic Way,” in Proceedings of the 14th International Conference in Web Engineering. In Press, 2014.

12.Referencias

[1] S. Ceri, P. Fraternali, M. Matera, Conceptual modeling of data-intensive Web applications, IEEE Internet Comput. 6 (2002).

[2] N. Koch, A. Knapp, G. Zhang, H. Baumeister, UML-Based Web Engineering, Springer, London, 2008.

[3] G. Rossi, O. Pastor, D. Schwabe, L. Olsina, Modeling and Implementing Web Applications using OOHDM, in: G. Rossi, O. Pastor, D. Schwabe, L. Olsina (Eds.), Web Eng. Model. Implement. Web Appl., Springer London, London, 2008: pp. 109–155.

[4] N. Koch, M. Pigerl, G. Zhang, T. Morozova, Patterns for the Model-Based Development of RIAs, in: 9th Int. Conf. Web Eng., Springer, Berlin, Heidelberg, 2009: pp. 283–291.

[5] M. Wimmer, A. Schauerhuber, H. Kargl, On the Integration of Web Modeling Languages: Preliminary Results and Future Challenges, in: Proc. 3rd Int. Work. Model. Web Eng., 2007.

[6] J.M. Rivero, G. Rossi, J. Grigera, E.R. Luna, A. Navarro, From Interface Mockups to Web Application Models, in: 12th Int. Conf. Web Inf. Syst. Eng., Sydney, Australia, 2011: pp. 257–264.

[7] A. Ravid, D.M. Berry, A Method for Extracting and Stating Software Requirements that a User Interface Prototype Contains, Requir. Eng. 5 (2000) 225–241.

[8] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, E. Astesiano, On the effectiveness of screen mockups in requirements engineering, in: 2010 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas., ACM Press, New York, NY, USA, 2010.

[9] Z. Huzar, L. Kuzniarz, G. Reggio, J.L. Sourrouille, Consistency Problems in UML-Based Software Development, Lect. Notes Comput. Sci. 3297 (2005) 1–12.

[10] VersionOne Inc., State of Agile Survey, 2011.

[11] L. Cao, B. Ramesh, Agile Requirements Engineering Practices: An Empirical Study, IEEE Softw. 25 (2008) 60–67.

[12] H. Ton, A Strategy for Balancing Business Value and Story Size, in: Agil. 2007 Conf., IEEE Computer Society, Washington, DC, USA, 2007: pp. 279–284.

[13] J. Ferreira, J. Noble, R. Biddle, Agile Development Iterations and UI Design., in: Agil. 2007 Conf., IEEE Computer Society, Washington, DC, 2007: pp. 50–58.

[14] A. Martin, R. Biddle, J. Noble, The XP Customer Role in Practice: Three Studies., in: Agil. Dev. Conf., IEEE Computer Society, Salt Lake City, Utah, USA, 2004: pp. 42–54.

[15] K.S. Mukasa, H. Kaindl, An Integration of Requirements and User Interface Specifications, in: 6th IEEE Int. Requir. Eng. Conf., IEEE Computer Society, Barcelona, Catalunya, Spain, 2008: pp. 327–328.

[16] M. Cohn, User stories applied: for agile software development, Addison-Wesley, 2004.

[17] K. Schwaber, Agile Project Management with Scrum, Microsoft Press, 2009.

[18] S. Ceri, P. Fraternali, A. Bongio, Web Modeling Language (WebML): a modeling language for designing Web sites, Comput. Networks. 33 (2000) 137–157.

[19] A. Rashid, D. Meder, J. Wiesenberger, A. Behm, Visual Requirement Specification In End-User Participation, in: First Int. Work. Multimed. Requir. Eng., IEEE Computer Society, Minnneapolis, MN, USA, 2006: pp. 6–6.

[20] K. Schneider, Generating fast feedback in requirements elicitation, in: Proc. 13th Int. Work. Conf. Requir. Eng. Found. Softw. Qual., 2007: pp. 160–174.

[21] J. Guerrero-Garcia, J.M. Gonzalez-Calleros, J. Vanderdonckt, J. Munoz-Arteaga, A Theoretical Survey of User Interface Description Languages: Preliminary Results, in: Proc. 2009 Lat. Am. Web Congr., IEEE, 2009: pp. 36–43.

[22] P.P. da Silva, User Interface Declarative Models and Development Environments: A Survey, Lect. Notes Comput. Sci. 1946 (2001) 207–226.

[23] J. Lin, M.W. Newman, J.I. Hong, J.A. Landay, DENIM: finding a tighter fit between tools and practice for Web site design, (2000) 510–517.

[24] A. Coyette, J. Vanderdonckt, Q. Limbourg, SketchiXML: A Design Tool for Informal User Interface Rapid Prototyping, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[25] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, V. López-Jaquero, USIXML: A Language Supporting Multi-path Development of User Interfaces, in: R. Bastide, P. Palanque, J. Roth (Eds.), Eng. Hum. Comput. Interact. Interact. Syst. Jt. Work. Conf. EHCI-DSVIS 2004, Springer Berlin Heidelberg, Hamburg, Germany, 2005: pp. 200–220.

[26] S. Kelly, J.-P. Tolvanen, Domain-Specific Modeling: Enabling Full Code Generation, Wiley-IEEE Computer Society, 2008.

[27] P. Duvall, Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley Professional, 2007.

[28] S.W. Ambler, Agile model driven development is good enough, IEEE Softw. 20 (2003) 71–73.

[29] C.S. LaRoche, B. Traynor, User-centered design (UCD) and technical communication: The inevitable marriage, in: 2010 IEEE Int. Prof. Comun. Conf., IEEE, 2010: pp. 113–116.

[30] ISO9241-210, Ergonomics of human-system interaction -- Part 210: Human-centred design for interactive systems, (2010).

[31] M. Maguire, Methods to Support Human-Centred Design, Int. J. Hum. Comput. Stud. 55 (2001) 587–634.

[32] T.Z. Warfel, Prototyping: A Practitioner's Guide, Rosenfeld Media, 2009.

[33] J.M. Rivero, J. Grigera, G. Rossi, E. Robles Luna, N. Koch, Improving Agility in Model-Driven Web Engineering, in: Proc. 23rd Int. Conf. Adv. Inf. Syst. Eng., CEUR, London, England, 2011: pp. 163–170.

[34] J.M. Rivero, J. Grigera, G. Rossi, E.R. Luna, N. Koch, Towards Agile Model-Driven Web Engineering, Lect. Notes Bus. Inf. Process. 107 (2012) 142–155.

[35] J.M. Rivero, E. Robles Luna, J. Grigera, G. Rossi, Improving user involvement through a model-driven requirements approach, in: 2013 Int. Work. Model. Requir. Eng. (MoDRE),, Rio de Janeiro, Brazil, 2013: pp. 20–29.

[36] J.M. Rivero, G. Rossi, J. Grigera, J. Burella, E. Robles Luna, S. Gordillo, From Mockups to User Interface Models: an Extensible Model Driven Approach, in: Proc. 10th Int. Conf. Web Eng., Springer-Verlag Berlin, Heidelberg, Vienna, Austria, 2010: pp. 13–24.

[37] E. Gamma, Design patterns: elements of reusable object-oriented software, Addison-Wesley, 1995.

[38] V. Basili, G. Caldiera, D. Rombach, The Goal Question Metric approach, (1994).

[39] M. van Welie, G. van der Veer, Pattern Languages in Interaction Design: Structure and Organization, in: Proc. Interact '03, 2003: pp. 527 – 534.

[40] G. Macbeth, E. Razumiejczyk, R.D. Ledesma, Cliff's Delta Calculator : A non-parametric effect size program for two groups of observations, Univ. Psychol. 10 (2011) 545–555.

[41] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, 2012.

[42] P. Rodríguez, A. Yagüe, P.P. Alarcón, J. Garbajosa, Some findings concerning requirements in Agile methodologies, Prod. Softw. Process Improv. 32 (2009) 171–184.

[43] J.M. Rivero, S. Heil, J. Grigera, M. Gaedke, G. Rossi, MockAPI: An Agile Approach Supporting API-first Web Application Development, in: Springer (Ed.), Proc. 13th Int. Conf. Web Eng., Aalborg, Denmark, 2013: pp. 7–21.

[44] B. Mulloy, Web API Design: Crafting Interfaces that Developers Love, Apigee, 2012.

[45] O. Chudnovskyy, M. Gaedke, Development of Web 2.0 Applications using WebComposition/Data Grid Service, in: Proc. Second Int. Conf. Adv. Serv. Comput., 2010: pp. 55–61.

[46] J.M. Rivero, S. Heil, J. Grigera, E. Robles Luna, M. Gaedke, An extensible, Model-Driven and end-user centric approach for API building, in: Proc. 14th Int. Conf. Web Eng. Press, Tolouse, France, 2014.

[47] G. Hohpe, B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional, 2003.

[48] J.I. Panach, S. España, I. Pederiva, O. Pastor, Capturing Interaction Requirements in a Model Transformation Technology Based on MDA., J. UCS. 14 (2008) 1480–1495.

[49] F. Paternò, ConcurTaskTrees: An Engineered Notation for Task Models, in: D. Diaper, N. Stanton (Eds.), Handb. Task Anal. Human-Computer Interact., Lawrence Erlbaum Associates, Manwah, 2003: pp. 483–503.

[50] E. Robles Luna, I. Garrigós, J. Grigera, M. Winckler, Capture and Evolution of Web Requirements Using WebSpec, in: B. Benatallah, F. Casati, G. Kappel, G. Rossi (Eds.), 10th Int. Conf. Web Eng., Springer Berlin Heidelberg, Vienna, Austria, 2010: pp. 173–188.

[51] U.B. Sangiorgi, S.D.J. Barbosa, MoLIC Designer: Towards Computational Support to HCI Design with MoLIC, in: Proc. 1st ACM SIGCHI Symp. Eng. Interact. Comput. Syst. - EICS '09, ACM Press, New York, New York, USA, 2009: p. 303.

[52] A. Homrighausen, H.-W. Six, M. Winter, Round-Trip Prototyping Based on Integrated Functional and User Interface Requirements Specifications, Requir. Eng. 7 (2002) 34–45.

[53] D. Kulak, E. Guiney, Use cases: requirements in context, Addison-Wesley, 2004.

[54] J.M. Moore, Communicating requirements using end-user GUI constructions with argumentation, in: 18th IEEE Int. Conf. Autom. Softw. Eng., IEEE Computer Society, Montreal, Quebec, Canada, 2003: pp. 360–363.

[55] R. Ramdoyal, A. Cleve, J.-L. Hainaut, Reverse Engineering User Interfaces for Interactive Database Conceptual Analysis, in: B. Pernici (Ed.), 22th Int. Conf. Adv. Inf. Syst. Eng., Springer Berlin Heidelberg, Hammamet, Tunisia, 2010: pp. 332–347.

[56] D. Rosenberg, M. Stephens, Collins-Cope, Agile Development with ICONIX Process—People, Process, and Pragmatism, 2005.

[57] M.H. Fortuna, C.M.L. Werner, M.R.S. Borges, Info Cases: Integrating Use Cases and Domain Models, in: 16th IEEE Int. Requir. Eng. Conf., IEEE Computer Society, Catalunya, Spain, 2008: pp. 81–84.

[58] L. Constantine, Canonical Abstract Prototypes for Abstract Visual and Interaction Design, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[59] J. Tidwell, Designing Interfaces: Patterns for Effective Interaction Design, O'Reilly Media, 2005.

[60] S. Pérez, F. Durao, S. Meliá, P. Dolog, O. Díaz, RESTful , Resource-Oriented Architectures : A Model-Driven Approach, in: Web Inf. Syst. Eng. – WISE 2010 Work., Springer, Hong Kong, China, 2010: pp. 282–294.

[61] F. Valverde, O. Pastor, Dealing with REST Services in Model-driven Web Engineering Methods, in: V Jornadas Científico-Técnicas En Serv. Web y SOA, JSWEB, 2009.

[62] D. Firmenich, S. Firmenich, J.M. Rivero, L. Antonelli, A Platform for Web Augmentation Requirements Specification, in: Proc. 14th Int. Conf. Web Eng. Press, Tolouse, France, 2014.