ESTRATEGIA HÍBRIDA BASADA EN CÓDIGO Y MODELOS PARA EL DESARROLLO ÁGIL DE APLICACIONES WEB

Autor: Karina Jimbo Pinos

Director: Dr. Gustavo Rossi.

Tesis presentada para obtener el grado de Magister en Ingeniería de Software.

FACULTAD DE INFORMÁTICA

UNIVERSIDAD NACIONAL DE LA PLATA

Febrero - 2015

3.1 RESUMEN

La metodología Mockup-Driven Development (MockupDD) propone utilizar prototipos de interfaz de usuario (usualmente conocidos como Mockups) como artefactos principales de elicitación de requerimientos y modelado en el contexto de un proceso Model-Driven Web Engineering (MDWE). Su proceso se basa en construir estos prototipos de manera obligatoria, temprana y de su posterior anotación al utilizar un microlenguaje de tags. MockupDD propone un enfoque ágil para MDWE, sin embargo, tratar conceptos del lenguaje para implementar requerimientos específicos sigue presentado las mismas dificultades que en los procesos MDWE convencionales: es necesario agregar nuevos elementos al metamodelo y modificar generadores de código.

En esta tesis se presenta una estrategia (extensión de MockupDD) que propone agregar características al paradigma MDWE ortodoxo, en donde, cada concepto del lenguaje en lugar de ser inmutable será ejecutable por sí mismo, convirtiéndose en prototipos de implementaciones por defecto, que pueden modificarse según se requiera al utilizar codificación manual – a costas de sacrificar parte o la totalidad de su abstracción. Esto provocará cambios importantes en comparación con la versión de MockupDD original, al plantear una nueva arquitectura de semántica de tags y un alejamiento del paradigma MDWE puro en pos de mejorar la agilidad y la adaptabilidad.

Palabras Claves: MDWE, Metodologías Ágiles, MockupDD, mockup, tag.

4.1

5.1 ABSTRACT

The Mockup - Driven Development (MockupDD) methodology proposes using user interface prototypes (usually known as Mockups) as key artifacts of requirements elicitation and modeling in the context of Model-Driven Web Engineering (MDWE) process. Its process is based on building these prototypes early and their subsequent annotation using a microlanguage of tags. MockupDD proposes an agile approach for MDWE. However, in this methodology extending language concepts presents the same difficulties as in conventional MDWE processes: you need to add new elements to the metamodel and modify code generators.

In this thesis proposes a strategy (an extension to MockupDD) that adds features to the orthodox MDWE paradigm. In this work, each concept of the language it is executable by itself instead of being immutable, thus becoming a prototype of a default implementation that can be modified as required using manual coding - at the expense of sacrificing some or all of its abstraction. This will cause major changes compared to the original version MockupDD, proposing a new architecture to specify tags semantics and a departure from the pure MDWE paradigm, towards improving agility and adaptability.

Keywords: MDWE, Agile methodologies, MockupDD, mockup, tag.

6.1 DEDICATORIA

A las personas más importantes de mi vida, por los sacrificios realizados para que pueda cumplir este sueño, por siempre creer en mí, ahora puedo decir que esta tesis lleva mucho de ustedes y de los momentos vividos que lograron hacernos una familia fuerte y unida.

Gracias por estar a mi lado, los AMO.

Mati, Jota, Markys

7.1 AGRADECIMIENTOS

A todos quienes hicieron posible culminar con mis estudios.

*En primer lugar agradezco a **Dios** por permitir cumplir esta meta.*

A mis padres, por siempre apoyarme incluso cuando al principio este sueño sonaba a locura. Sin ustedes no hubiera podido culminar esta etapa.

*A mis hermanos, que hicieron que me sintiera en casa a pesar de la distancia, en especial a **Juan Pablo** por tu tiempo y apoyo incondicional.*

A mis suegros, cuñados(as), mis sobrinos porque nunca faltó un saludo y frases de apoyo.

A la Universidad Nacional de la Plata, y a todos aquellos maestros que me brindaron su conocimiento para mi preparación. De manera especial al Dr. Gustavo Rossi, mi director de tesis que supo guiarme en desarrollo de este trabajo.

*A **Matías Rivero**, quien me apoyó desde el principio hasta el final en la elaboración de mi tesis, por su tiempo, paciencia para trasmitirme sus conocimientos y sus palabras de aliento.*

*A la **SENESCYT**, institución que confió en mí, al proporcionarme la beca de estudios de cuarto nivel.*

A la familia Gomez - Degue, gracias por haber hecho de nuestra estadía en Argentina un tiempo para recordar con una sonrisa, siempre estarán en nuestros pensamientos.

Gracias!

8.1_____

CONTENIDO

3.1RESUMEN.....2

5.1ABSTRACT.....3

6.1DEDICATORIA.....4

7.1AGRADECIMIENTOS.....5

8.1CONTENIDO.....6

9.1LISTA DE FIGURAS.....10

10.1LISTA DE TABLAS.....11

12.1CAPÍTULO 1. INTRODUCCIÓN 12

1.1Justificación de la Investigación.....12

1.2Objetivo de la tesis.....14

1.2.1Objetivo general.....14

1.2.2Objetivos específicos.....14

1.3Metodología de la Investigación.....14

1.4Organización de la tesis.....15

13.1CAPÍTULO 2. ESTADO DEL ARTE.....17

2.1.MDD: Model Driven Development.....18

2.1.1.Ventajas de MDD.....18

2.1.2.Obstáculos que enfrenta MDD.....20

2.1.3.MDA: Model Driven Architecture.....20

2.1.4.AMDD: Agile Model Driven Development.....20

2.1.5.MDWE: Model Driven Web Engineering.....21

2.1.6.Metodologías MDWE 21

2.1.6.1.OOHDM: Object-Oriented Hypermedia Design Method 22

2.1.6.2.UWE: UML-Based Web Engineering.....24

2.1.6.3.WebML: Web Modeling Language.....26

2.2.Métodos ágiles.....27
2.2.1.Manifiesto ágil.....29
2.2.1.1.Principios.....30
2.2.1.2.Scrum.....31
2.2.1.2.1.Principales elementos.....32
2.2.1.2.2.Proceso.....33
2.2.2.Proceso de Desarrollo Ágil para la Web 35
2.3.MockupDD: Modelos + Agilidad.....35
2.3.1.Beneficios del uso mockups.....36
2.3.2.Proceso.....37
2.3.3.Proceso de desarrollo 41
2.3.4.Herramientas de soporte.....41
14.1CAPÍTULO 3. ESTRATEGIA PROPUESTA.....43
3.1.Introducción 43
3.2.Tipos de proyectos.....44
3.3.Aplicación Ejemplo.....44
3.4.Proceso de desarrollo.....44
3.4.1.Paso 1 y 2. Construcción de Mockups y procesamiento.....45
3.4.2.Paso 3. Especificación de características.....48
3.4.3.Conjunto de tags básico.....49
3.4.3.1.ListTag.....49
3.4.3.2.PropertyListTag.....50
3.4.3.3.DeleteListTag.....50
3.4.3.4.DataEntryTag.....51
3.4.3.5.DataPropertyTag.....51
3.4.3.6.SaveTag.....52

3.4.3.7.DeleteTag....52
3.4.3.8.LinkTag....53
3.4.3.9.TransferTag....53
3.4.3.10.DataTag....54
3.4.3.11.DataNavTag....54
3.4.4.Paso 4. Generación de código....55
3.5.Valor Agregado....55
3.6. Arquitectura....56
14.1Ejecución de tags en la plataforma....56
14.2Lado Cliente....56
14.3Lado Servidor....57
14.4Guardar tags para un mockup....57
16.1CAPÍTULO 4. PROTOTIPO....58
4.1. Introducción....58
4.2. Visión del Prototipo....58
16.1Alcance....58
4.3.Metodología....59
4.4. Herramientas utilizadas....59
4.5.Diseño....62
4.5.1.Diagrama de Clases....62
4.5.2.Diccionario de Clases....63
4.6.Implementación....64
4.6.1.Tag - Lado Cliente....64
4.6.2.Tag - Lado Servidor....66
4.6.3.Herramienta de Administración de Tags....66
4.6.3.1.TaggingTool....67
4.6.3.2.Client Side....67

4.6.3.3.Server Side....68
4.6.4.Guardar tags para un mockup....68
4.6.5.Herramienta de Etiquetado....69
4.6.6.Herramienta de Ejecución....72
4.7.Pruebas 73
16.2Manejo de Tags: Extensión / modificación de tags existentes....82
16.3 Discusión del prototipo....86
17.1CAPÍTULO 5. MEDICIÓN DE LA HERRAMIENTA....88
5.1 Opinión de un conjunto de desarrolladores....88
5.1.1Diseño del cuestionario....88
5.1.2Cuestionario de evaluación MetapiTag....89
5.1.3Análisis de los resultados....91
5.1.3.1Resultados agrupados por característica....91
5.1.3.2Resultados Finales....92
5.2 Creación de un nuevo elemento de lenguaje (Comparación WebRatio -MatapiTag)....93
5.2.1Creación cliente HTTP 94
5.2.1.1Cliente HTTP - WebRatio....94
5.2.1.2ClienteHttpTag - MetapiTag....98
5.2.2Creación PostSocialUnit....101
5.2.2.1SocialUnitPost - WebRatio....101
5.2.2.2SocialPostTag – MetapiTag....102
5.2.3Resumen. Creación unidad personalizada WebRatio....102
5.2.4Resumen. Creación tag personalizado MetapiTag....103
5.2.5Análisis de los resultados....104
19.1CAPÍTULO 6. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO....105
6.1 Verificación y evaluación de los objetivos....105

6.2 Aportaciones de la Tesis.....106
6.3 Futuras líneas de trabajo.....107
20.1ACRÓNIMOS.....108
22.1BIBLIOGRAFÍA.....109
23.1ANEXO A. CÓDIGO DE LOS TAGS.....112

9.1 LISTA DE FIGURAS

10.1__LISTA DE TABLAS

11.1__

12.1 CAPÍTULO 1. INTRODUCCIÓN

En este capítulo, se describe la motivación y el contexto en el que se enmarca esta tesis; los problemas y los objetivos planteados en el presente trabajo de investigación; la metodología a seguir; una breve mención de los temas que se tratarán y la organización general del documento.

1.1 Justificación de la Investigación

Metodologías ágiles como Scrum [1] promueven la interacción temprana y constante con los clientes para afirmar que el software pactado cumple con los requerimientos solicitados, mediante la entrega constante de prototipos desarrollados en períodos cortos de tiempo. Estos prototipos alimentan de forma continua el desarrollo y permiten validar, capturar y descubrir requerimientos con rapidez.

Las metodologías MDWE (Model Driven Web Engineering) [2] tienen un enfoque top-down o en cascada y promueven la creación de aplicaciones web mediante la construcción de un conjunto de modelos en diferentes niveles de abstracción:

Modelo de Dominio: define los objetos de dominio y sus relaciones.

Modelo de hipertexto: define los nodos de navegación y sus enlaces.

Modelo de presentación: describe en detalle aspectos de presentación especificados en el modelo de hipertexto

Mientras que las metodologías MDWE facilitan la especificación de software al elevar el nivel de las abstracciones (lo cual incrementa la productividad), fallan en proveer una interacción ágil con los clientes ya que los resultados concretos se obtienen demasiado tarde.

Por otro lado, mientras que esta característica es provista por las metodologías ágiles, las mismas están basadas en codificación directa y no proporcionan abstracción, portabilidad y productividad a través de la generación automática de código que las metodologías MDWE proveen.

Implementar requerimientos que exceden las capacidades expresivas en el contexto de las metodologías MDWE implica tener que modificar su infraestructura, al cambiar su metamodelo y generadores de código, lo cual resulta costoso y poco práctico, además de requerir experiencia y conocimientos específicos del equipo de desarrollo. Las Metodologías ágiles tradicionales evitan este problema al utilizar codificación directa y sacrifican la productividad asociada al uso de especificaciones más abstractas.

Mockup-Driven Development [3] (MockupDD) propone un enfoque ágil para MDWE, parte de mockups que facilitan la generación de prototipos y modelos de software, como resultado, se obtiene un proceso iterativo ágil con ventajas de uno MDWE. Sin embargo, tratar conceptos del lenguaje para implementar requerimientos específicos sigue presentado las mismas dificultades que en los procesos MDWE convencionales: es necesario agregar nuevos elementos al metamodelo y modificar generadores de código.

Con el fin de superar dificultades inherentes al paradigma MDWE clásico que presenta MockupDD, surge el interés y motivación de este trabajo. Se propone mejorar MockupDD, al agregar características al paradigma MDWE ortodoxo que la misma metodología sigue, con el objeto de permitir modificar la semántica de los conceptos de su lenguaje, dependiendo del dominio específico.

En este contexto, en lugar de ser sólo elementos estáticos del lenguaje, cada tag[1] pasará a ser “ejecutable” por sí mismo, al formar prototipos de implementaciones por defecto en el lenguaje, que pueden modificarse según se requiera – a costas de sacrificar parte o la totalidad de su abstracción. Los elementos modificados se podrán incluir como nuevas especificaciones, y mejorarán el lenguaje.

El diseño de la arquitectura tratará a cada elemento de lenguaje como prototipos de implementaciones y no como conceptos estáticos e inmutables, lo que provocará cambios importantes en comparación con la versión MockupDD

1 Un tag es una anotación formal con una gramática concreta, permite declarar diferentes especificaciones relativas a determinados aspectos de la aplicación desarrollada (persistencia, navegación, validación) en pos de su rápida implementación.

original y un posible alejamiento del paradigma MDWE puro en pos de mejorar la agilidad y adaptabilidad de la metodología.

1.2 Objetivo de la tesis

1.2.1 Objetivo general

Proponer una estrategia de mejora a MockupDD, al agregar características al paradigma MDWE ortodoxo que posee un enfoque cascada mediante la construcción de modelos en diferentes niveles de abstracción, que permita modificar la semántica de los conceptos de su lenguaje dependiendo de un dominio específico.

1.2.2 Objetivos específicos

- Realizar un estudio del estado del arte en relación a metodologías ágiles, MDWE y MockupDD para determinar el estado actual y ahondar en las limitaciones que presentan.
- Plantear una arquitectura que permita superar las dificultades inherentes al paradigma MDWE puro de MockupDD, al tratar los elementos de lenguaje como prototipos de implementación y no como conceptos estáticos e inmutables.
- Implementar un prototipo enmarcado dentro de la arquitectura propuesta.
- Evaluar la estrategia propuesta al validar el funcionamiento del prototipo, tomando en cuenta variables como tiempos de desarrollo, aprendizaje y ajuste a requerimientos específicos.

1.3 Metodología de la Investigación

La investigación se llevará a cabo dentro de un desarrollo tecnológico, a continuación se detalla en líneas generales los principales pasos a seguir:

- Determinación del problema
    - Análisis y delimitación del problema a abordar.
- Búsqueda de documentación

- Para construir el conocimiento asociado al presente proyecto, se analizará información de la situación actual del desarrollo dirigido por modelos, metodologías ágiles y MockupDD.

- Desarrollo de la Investigación
  - Determinar a qué tipo de aplicaciones va dirigida la propuesta, que etapas del ciclo de vida de software se abordarán.
  - Crear una estrategia que permita superar las limitaciones encontradas.
  - Implementar un prototipo enmarcado en la estrategia diseñada, que permita validar el cumplimiento de los objetivos planteados mediante la utilización de la misma en un caso práctico.
  - Efectuar la revisión del diseño, arquitectura a través de pruebas sobre prototipo.
- Llevar a cabo evaluaciones
  - Recolectar opiniones de un conjunto de desarrolladores, posterior a una demostración de la herramienta.
  - Comprobar que se ha alcanzado el objetivo previsto, al darle más versatilidad a MockupDD, al agregar nuevos elementos de lenguaje tanto en la herramienta propuesta como en una herramienta del mercado basada en modelos.
- Análisis de resultados y elaboración de conclusiones
  - Determinar en qué medida se han cumplido los objetivos planteados al solucionar los problemas encontrados al principio de la investigación y analizar si es posible realizar mejoras.

1.4 Organización de la tesis

En consonancia con la metodología descrita (Metodología de la Investigación), la memoria de tesis se construye sobre la siguiente estructura.

Capítulo 1. Exhibe una introducción, en la que se indica la motivación del trabajo de investigación, planteamiento del problema, los objetivos que se persiguen, la metodología a seguir y la estructura del documento.

Capítulo 2. Presenta una recopilación de las metodológicas que servirán de base para la investigación, así se hace una revisión de los principales aspectos de las metodologías dirigidas por modelos, metodologías ágiles, MockupDD y como los prototipos contribuyen a la correcta recopilación de requerimientos.

Capítulo 3. En este capítulo se describe una estrategia, que permite superar las dificultades planteadas. Se detalla la arquitectura de los elementos de lenguaje.

Capítulo 4. En este capítulo se describe la construcción del prototipo basado en la estrategia propuesta, y se realizan las pruebas necesarias para validar el funcionamiento del mismo.

Capítulo 5. En este capítulo se evalúan los objetivos plateados. Mediante la recopilación de opiniones de un conjunto de desarrolladores previo la demostración del prototipo y mediante la comparación de un conjunto de métricas, basadas en la construcción de nuevos elementos de lenguaje en el prototipo construido y en otra herramienta basada en modelos.

Capítulo 6. En este capítulo se presentan las conclusiones del trabajo realizado. Se detallan los aportes, limitaciones y posibles extensiones que surgen de la investigación.

Finalmente en los anexos se detallan todas aquellas cuestiones que se han tratado y facilitan el entendimiento de este trabajo.

13.1 CAPÍTULO 2. ESTADO DEL ARTE

En la sociedad actual la demanda de las aplicaciones tecnológicas es cada vez más elevada, y presenta nuevos desafíos como tiempos de construcción reducidos y requerimientos volátiles, si bien el proceso de desarrollo de software ha evolucionado, no logra ajustarse al crecimiento exponencial de las demandas surgidas, para hacer frente a estos retos han nacido varias metodologías con sus pros y contras.

El desarrollo de software dirigido por modelos MDD (Model Driven Development), promueve la reutilización y reduce los esfuerzos de construcción de software, a partir de modelos de alto nivel, alejándose de la codificación directa. Los modelos representan los aspectos del programa final ejecutable, por lo que deben ser formales, precisos y con una semántica bien definida. La iniciativa MDA[4] y los estándares que engloba tienen como objetivo situar los modelos en un nivel de abstracción tal que su realización en múltiples plataformas sea posible.

El desarrollo ágil define nuevos métodos para afrontar la construcción del software de una forma más eficiente, y por tanto menos costosa. Estos métodos en contraposición a los tradicionales, asumen el cambio como algo inevitable, para lo cual se hace necesaria una nueva forma de abordar el desarrollo que facilite el acomodo a los nuevos requisitos a medida que éstos surjan, en vez de pretender analizar inicialmente el dominio a modelar de forma tan exhaustiva que ya no se produzca luego ningún cambio o estos sean mínimos. Estos métodos ligeros están centrados en el código el mismo que se convierte en la principal documentación del proyecto.[5]

A continuación se profundiza en estas dos metodologías, que servirán de base durante la investigación.

2.1. MDD: Model Driven Development

El desarrollo de software dirigido por modelos se ha convertido en un nuevo paradigma de construcción software y promete mejorar el proceso basándose en modelos.

El adjetivo “dirigido” (driven) en MDD enfatiza que el paradigma asigna a los modelos un rol central y activo; son al menos tan importantes como el código fuente. Los modelos se generan desde los más abstractos a los más concretos a través de pasos de transformación y/o refinamientos, hasta llegar al código al aplicar una última transformación [6], este modelo se ejecuta directamente por una plataforma que cumple con los requerimientos, incluidos los no funcionales[7]. MDD se basa en un lenguaje particular cercano al dominio del problema a resolver y se le conoce como metamodelo.

MDD se refiere a las transformaciones en etapas del desarrollo, por ejemplo de la especificación de requerimientos a modelos de diseños y de estos últimos a implementaciones, estas transformaciones entre modelos proporcionan una cadena que permite la implementación de un sistema automatizado en sucesivos pasos de los diferentes modelos.

2.1.1. Ventajas de MDD

El desarrollo de software al estar guiado por modelos produce los siguientes beneficios:[6][8]

Incremento en la productividad: MDD reduce los costos de desarrollo de software mediante la generación automática del código y otros artefactos a partir de los modelos, lo cual incrementa la productividad de los desarrolladores, permitiendo que el trabajo lo realice la herramienta y no el programador.

Portabilidad: el progreso de la tecnología hace que los componentes de software se vuelvan obsoletos, MDD ayuda a solucionar este problema a través de una arquitectura fácil de mantener donde los cambios se implementan rápida y de forma consistente, al habilitar una migración eficiente de los componentes hacia las nuevas tecnologías. Los modelos de alto nivel están libres de detalles de implementación y facilitan la adaptación a los cambios que pueda sufrir la

plataforma tecnológica subyacente o la arquitectura de implementación. Lo anterior es posible siempre que se implementen nuevos generadores de código para las nuevas plataformas.

Consistencia: la aplicación manual de las prácticas de codificación y diseño es una tarea propensa a errores. A través de la automatización MDD favorece la generación consistente de los artefactos.

Re-uso: en MDD se invierte en el desarrollo de modelos y transformaciones. Esta inversión se amortiza a medida que los modelos y las transformaciones son reusados. Por otra parte el reúso de artefactos ya probados incrementa la confianza en el desarrollo de nuevas funcionalidades y reduce los riesgos ya que los temas técnicos han sido previamente resueltos.

Mejoras en la comunicación con los usuarios: los modelos omiten detalles de implementación que no son relevantes para entender el comportamiento lógico del sistema. Por ello están más cerca del dominio del problema, reduciendo la brecha semántica entre los conceptos que son entendidos por los usuarios y el lenguaje en el cual se expresa la solución. Esta mejora en la comunicación influye favorablemente en la producción de software, alineándolo con los objetivos de sus usuarios.

Mejoras en la comunicación entre los desarrolladores: los modelos facilitan el entendimiento del sistema por parte de los distintos desarrolladores. Esto da origen a discusiones más productivas y permite mejorar los diseños. Además, el hecho de que los modelos son parte del sistema y no sólo documentación, hace que permanezcan actualizados y confiables.

Captura de la experiencia: las organizaciones y los proyectos frecuentemente dependen de expertos clave quienes toman las decisiones respecto al sistema. Al capturar su experiencia en los modelos y en las transformaciones, otros miembros del equipo pueden aprovecharla sin requerir su presencia. Además este conocimiento se mantiene aun cuando los expertos se alejen de la organización.

Los modelos son productos de larga duración: en MDD los modelos son productos importantes que capturan lo que el sistema informático de la organización hace. Los modelos de alto nivel son resistentes a los cambios de plataforma y sólo sufren cambios cuando lo hacen los requisitos del negocio.

2.1.2. Obstáculos que enfrenta MDD

Un problema que afecta a muchos productos de software actuales es la usabilidad. En el caso de MDD la usabilidad se pone de manifiesto generalmente en sus herramientas que tienden a ser difíciles de aprender y de utilizar. Esta falencia se debe a la falta de experiencia que se posee sobre el uso de este tipo de herramientas. Un segundo problema técnico importante es que todavía hay muy poca base teórica para MDD. Mucha de la tecnología MDD disponible ha sido desarrollado de forma ad hoc por equipos profesionales al tratar de resolver problemas específicos. La falta de una teoría sólida se manifiesta también en los problemas de interoperabilidad entre las herramientas.

2.1.3. MDA: Model Driven Architecture

MDA surge como una propuesta de estandarización del enfoque MDD, adopta a los modelos como elementos esenciales de la cadena de producción de software, proporciona un conjunto de lenguajes estándares definidos por el OMG (*Object Management Group*) [9] donde los modelos son componentes esenciales en el proceso de desarrollo de software, el núcleo de la arquitectura está formada por estos estándares .

MDA define tres tipos de modelos. Los CIM, Computation Independent Model, asociados al dominio del negocio, los PIM, Platform Independent Model, asociados a modelos abstractos del software, y los PSM, Platform Specific Model, relacionados con modelos de software específicos de plataformas tecnológicas.

2.1.4. AMDD: Agile Model Driven Development

AMDD[10] es la versión ágil de MDD, crea modelos ágiles, en lugar de crear modelos extensos, es menos formal en el sentido que no necesariamente hay metamodelo ni generadores de código. Su objetivo es describir cómo

desarrolladores y stakeholders pueden trabajar juntos en colaboración para crear modelos apenas lo suficientemente buenos. No requiere que cada uno sea un experto en el modelado, sólo requiere que estén dispuestos a probar. Permite utilizar la herramienta de modelado más adecuada para el trabajo, a menudo herramientas muy simples, como las pizarras digitales o en papel, que sirvan para comunicarse de manera efectiva y no documentar exhaustivamente.[11]

AMDD permite a los desarrolladores pensar en cuestiones más amplias antes de sumergirse en los detalles de implementación. Un modelo ágil es un modelo que es apenas lo suficientemente bueno para la situación en cuestión y presentan los siguientes rasgos:

- Cumple su propósito.
- Comprensible.
- Lo suficiente preciso.
- Lo suficiente consistente.
- Lo suficiente detallado.
- Proporciona un valor positivo.
- Tan simple como sea posible.

2.1.5. MDWE: Model Driven Web Engineering

Desde el punto de vista de la ingeniería de software el desarrollo de aplicaciones web representa un nuevo dominio de aplicación [12]. MDWE es un enfoque basado en modelos para aplicaciones web, proporciona metodologías y herramientas para el diseño y desarrollo de este tipo de software. Utiliza modelos separados (navegación, presentación, datos) y se apoya en generadores de código, que producen la mayor parte de las páginas de la aplicación y lógica basada en los modelos.

2.1.6. Metodologías MDWE

Las características especiales de las aplicaciones web hacen surgir la necesidad de una adaptación de los enfoques de desarrollo existentes o quizás

el desarrollo de enfoques metodológicos nuevos. La construcción de aplicaciones web contempla varias características que difieren respecto del desarrollo de otras que no lo son. Por ejemplo:

- Diferentes tipos de stakeholders involucrados en el proceso de desarrollo (cliente, usuarios, analistas, diseñadores gráficos, desarrolladores, expertos en multimedia y seguridad, marketing)
- Estructura de navegación clara e intuitiva para el usuario.
- Interfaz gráfica de usuario, cuyo diseño tiene en cuenta, aspectos de marketing y multimedia.

La comunidad de Ingeniería Web aboga por el uso de modelos, con el fin de mejorar el proceso de desarrollo de aplicaciones Web. Las metodologías MDWE (Model Driven Web Engineering) [2][13] tienen un enfoque top-down o en cascada y promueven el desarrollo web mediante la construcción de un conjunto de modelos en diferentes niveles de abstracción:

Modelo de Dominio: define los objetos de dominio y sus relaciones.

Modelo de Hipertexto: define los nodos de navegación y sus enlaces.

Modelo de Presentación: describe en detalle aspectos de presentación especificados en el modelo de hipertexto

En la actualidad existen varias metodologías MDWE a continuación se hace una revisión de un subconjunto de enfoques metodológicos basados en modelos que pueden aportar conocimiento para el desarrollo del presente trabajo

2.1.6.1. OOHDM: Object-Oriented Hypermedia Design Method

OOHDM es un enfoque basado en modelos para desarrollar aplicaciones web. La especificación es vista como una instancia de un modelo hipermedia que especifica la semántica de navegación, a través del uso de varios meta-modelos especializados. Cada modelo se centra en diferentes aspectos de la aplicación. Una vez que los modelos se han especificado, es posible generar en tiempo de ejecución código que implementa la aplicación. [7]

En OOHDM una aplicación web se construye en un proceso de cuatro actividades con el apoyo de un modelo de proceso incremental o prototipo.

- **Diseño Conceptual**

Utiliza principios bien conocidos de modelado orientado a objetos. No se preocupa de los tipos de los usuarios y las tareas, sólo de la semántica del dominio de la aplicación. Un esquema conceptual se construye fuera de los subsistemas, clases y relaciones.

- **Diseño de navegación**

En OOHDM, una aplicación es considerada como una vista de navegación sobre el modelo conceptual. Esto refleja una importante innovación de OOHDM, que reconoce que los objetos de navegación de usuario no son los objetos conceptuales, pero son otros tipos de objetos que se han construido a partir de uno o más objetos conceptuales, para adaptarse a los usuarios y tareas que debe ser apoyado. En otras palabras, para cada perfil de usuario se puede definir una diferente estructura de navegación que refleja los objetos y las relaciones en el esquema conceptual de acuerdo a las tareas que un tipo de usuario realiza.

La estructura de clases de navegación de una aplicación web se define mediante un esquema las contiene. En OOHDM existe un conjunto predefinido de tipos básicos de las clases de navegación: nodos, links, anchors, y el acceso a estructuras. Diferentes aplicaciones (en el mismo dominio) pueden contener diferentes topologías de enlaces, de acuerdo con el perfil de los diferentes usuarios.

- **Diseño abstracto de interfaz**

Se construye mediante la definición de los objetos perceptibles también llamados widgets que contiene información en términos de clases de interfaz. Los objetos de interfaz se mapean a objetos de navegación, proporcionándoles un aspecto perceptible, para los valores de entrada. El comportamiento de la interfaz se define al especificar cómo controlar eventos externos o generados por

los usuarios y cómo la comunicación se lleva a cabo entre la interfaz y los objetos de navegación.

- **Implementación**

Con el modelo conceptual, de navegación y el abstracto de interfaz, sólo queda llevar los objetos a un lenguaje concreto de programación, para obtener así la implementación ejecutable.

2.1.6.2. UWE: UML-Based Web Engineering

UWE [14] es un enfoque de ingeniería de software para el dominio web que cubre todo el ciclo de vida de desarrollo.

Propuesto por Koch y Wirsing años 2000-2001, está basado en el proceso de desarrollo unificado [15] y el lenguaje de modelado UML (Unified Modeling Language)[16] ambos adaptados para aplicaciones web. Está totalmente basado en estándares y es una de sus principales ventajas.

Características

- Se centra en el usuario, en la etapa de captura de requerimientos y separa actividades de elicitación, especificación y validación.
- Orientado hacia aplicaciones web personalizadas o adaptativas.
- Utiliza UML extendiéndolo a través de un profile, para generar los modelos propuestos en el proceso.
- El proceso está más enfocado a actividades de análisis y de diseño, utiliza el paradigma de orientación a objetos.
- Los modelos son refinados en las iteraciones que plantea el proceso de desarrollo.
- Existen herramientas case como ArgoUML[17] y MagicUWE[18] que soporta UWE.
- Está compuesto por 3 fases o etapas:
    - Captura de requerimientos
    - Análisis y diseño

- Implementación

Captura de requerimientos: El objetivo de esta etapa es identificar los requerimientos que la aplicación web debe satisfacer y representarlos a través de un modelo de Casos de Uso [19]. Dado que es una técnica centrada en el usuario define los actores que van a interactuar con la aplicación y la funcionalidad que la aplicación web deberá cumplir para cada uno de ellos.

Análisis y diseño: toma como entrada el modelo de Casos de Uso a fin de generar:

- *El modelo conceptual del dominio:* un modelo de clases UML que contiene los atributos, operaciones de cada clase, las relaciones de asociación, dependencia, jerarquía entre las clases del modelo; identifica interfaces y restricciones.
- *El modelo de navegación:* se basa modelo conceptual. Es un modelo que muestra un diagrama de clases UML que representa las clases de navegación y las asociaciones entre estas.
- *El modelo de estructura de navegación:* muestra como el usuario navegará a través de las clases definidas en el modelo de navegación. Este modelo sirve de guía para construir el modelo de presentación.
- *El modelo de presentación:* Se enfoca en visualizar como es la organización estructural no en la apariencia física. Hay muchas posibilidades de construir un modelo de presentación, sobre un modelo de estructura de navegación dado. El resultado es un diagrama UML de clases construidas con estereotipos UML para elementos de presentación.

Implementación consiste en implementar los modelos validados e integrarlos. Se implementan desde los prototipos de interfaz de usuario hasta el código.

2.1.6.3. WebML: Web Modeling Language

WebML[20] (Ceri et al., 2000) es un lenguaje modelado de alto nivel para la especificación de aplicaciones web. Este enfoque considera cuatro características plasmadas en diferentes modelos: modelo estructural, modelo de hipertexto, modelo de presentación y modelo de personalización. WebML define también un proceso iterativo, con las siguientes etapas: recolección de requisitos, diseño de datos, diseño del hipertexto, diseño de presentación, diseño de usuarios y de grupos y diseño de personalización

Características

- Permite la descripción en alto nivel de una aplicación web, bajo perspectivas ortogonales, tales como: el contexto de datos, las páginas que la componen, los links o enlaces entre las páginas, la presentación gráfica por página y la personalización por grupo de usuarios.
- Cubre aspectos avanzados del modelado de aplicaciones web incluyendo presentación, modelado de usuarios y personalización.
- Define un proceso iterativo compuesto por fases que se repiten en cada uno de los ciclos de diseño.
    - *Especificación de requerimientos* de la aplicación tales como, usuarios a los que está dirigida, ejemplos de contenido, guías de estilo, personalización requerida, restricciones debido a datos heredados, entre otras.
    - *Diseño de datos* se identifican entidades y relaciones. WebML permite la utilización en esta etapa de modelos E- R (Entity-Relationship) o modelos de clases UML.
    - *Diseño del hipertexto* se basa en la construcción de dos modelos: el modelo de composición, en el que se identifican qué páginas componen el hipertexto y con qué contenido; y el modelo de navegación que expresa cómo se enlazan las páginas para formar el hipertexto.

- Las fases siguientes son la *implementación* de la aplicación que es automática, *pruebas y evaluación*, con el fin de mejorar su calidad interna y externa, *despliegue* de la arquitectura seleccionada, el *mantenimiento* y, de manera opcional la *evolución* de la aplicación una vez que se ha desplegado.

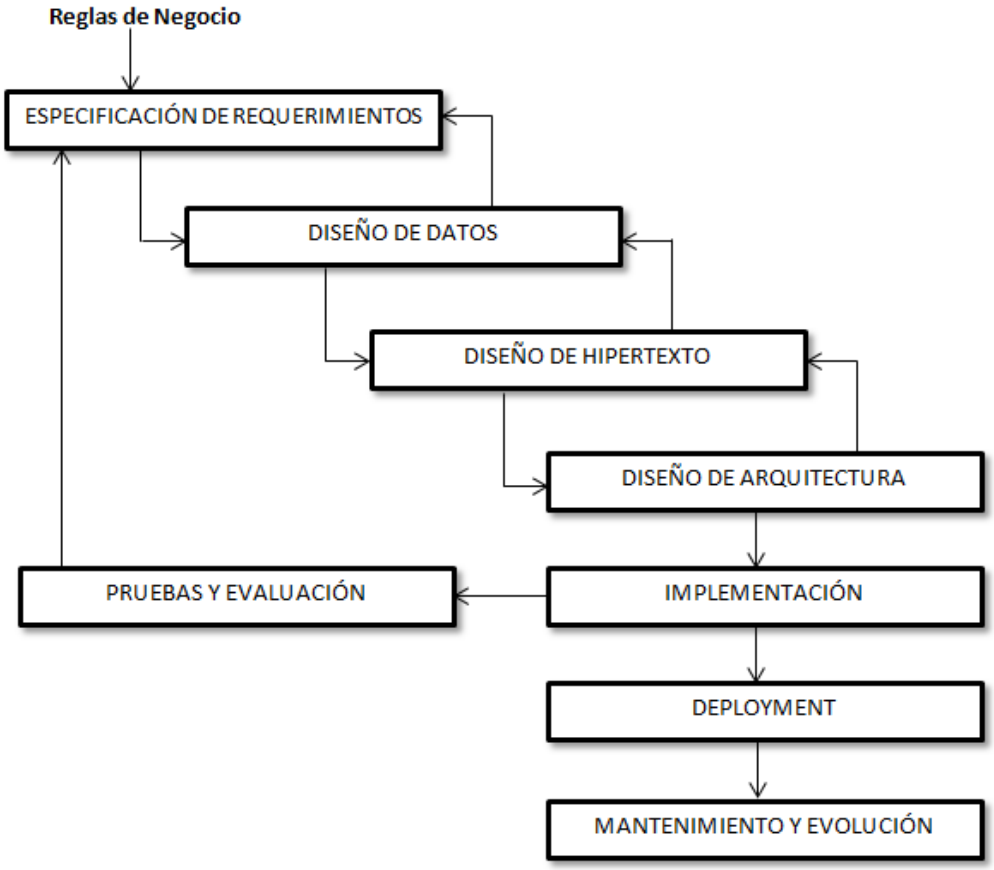


Figura 1. Fases en el proceso de desarrollo WebML[20]

En el 2013 surge IFML (Interaction Flow Modeling Language), el nuevo estándar adoptado por la OMG para modelado de interfaces de Usuario, basado en WebML.

2.2. Métodos ágiles

"Desde hace unos pocos años ha habido un interés creciente en las metodologías ágiles. Caracterizadas alternativamente como antídoto a la burocracia, han suscitado interés en el panorama del software." Martin Fowler.

Las metodologías tradicionales imponen un proceso disciplinado sobre el desarrollo y generan gran cantidad de documentación con el objetivo de facilitar y compartir el conocimiento entre los integrantes del equipo de trabajo, han estado presentes durante mucho tiempo y han demostrado ser efectivas, particularmente en lo que respecta a la administración de recursos a utilizar y a la planificación de los tiempos de desarrollo, en proyectos de gran tamaño con requerimientos estables. La crítica más frecuente a estas metodologías es que son burocráticas.

El origen de estas metodologías fue motivado por el desarrollo de proyectos donde los requerimientos del sistema son desconocidos, inestables o muy cambiantes, los tiempos de desarrollo se reducen drásticamente, y al mismo tiempo, se espera la producción de un producto de alta calidad, que a su vez garantice el mínimo riesgo ante la necesidad de introducir cambios a los requerimientos. Como una reacción a estas metodologías tradicionales, estructuradas, estrictas y sobrecargadas de técnicas y herramientas, ha surgido un nuevo grupo de metodologías desde los años 90. Durante algún tiempo se conocían como las metodologías ligeras o livianas, pero el término aceptado y definido por la Agile Alliance[21] es *Metodologías Ágiles (MA).*

Las MA han demostrado[22] afrontar el desarrollo del software de una forma más eficiente, y por tanto menos costosa. Estos métodos al contrario de los tradicionales, asumen el cambio como algo inevitable, para lo cual se hace necesaria una nueva forma de abordar el desarrollo de software que facilite el acomodo a los nuevos requerimientos a medida que éstos surjan, en vez de pretender analizar inicialmente el dominio a modelar de forma tan exhaustiva que ya no se produzca luego ningún cambio o estos sean mínimos.

Estos métodos ligeros están centrados en código, de tal forma que este se convierte en la principal documentación del proyecto. La ausencia de documentación pone de manifiesto dos diferencias mucho más significativas [23]:

Los métodos ágiles son adaptables, más que predictivos.

Los métodos ágiles se centran más en las personas que en el proceso.

2.2.1. Manifiesto ágil

El Manifiesto Ágil [5] es el documento que define los principios rectores de las metodologías ágiles de desarrollo de software, incluye cuatro postulados y una serie de principios asociados.

Postulados

1) Valorar al individuo y a las interacciones del equipo de desarrollo por encima del proceso y las herramientas. Tres premisas sustentan este postulado:

   a) los integrantes del equipo son el factor principal de éxito;

   b) es más importante construir el equipo de trabajo que construir el entorno;

   c) es mejor crear el equipo y que éste configure el entorno al tomar en cuenta sus necesidades.[24]

2) Valorar el desarrollo de software que funcione por sobre una documentación exhaustiva. El postulado se basa en la premisa que los documentos no pueden sustituir ni ofrecer el valor agregado que se logra con la comunicación directa entre las personas a través de la interacción con los prototipos. Se debe reducir al mínimo indispensable el uso de documentación que genera trabajo y que no aporta un valor directo al producto[24].

3) Valorar la colaboración con el cliente por sobre la negociación contractual. En el desarrollo ágil el cliente se integra y colabora con el equipo de trabajo. Se asume que el contrato en sí, no aporta valor al producto, sino que es sólo un formalismo que establece líneas de responsabilidad entre las partes.

4) Valorar la respuesta al cambio por sobre el seguimiento de un plan. La evolución rápida y continua deben ser factores inherentes al proceso de desarrollo. Se valora la capacidad de respuesta ante los cambios por sobre la capacidad de seguimiento y aseguramiento de planes pre-establecidos.

2.2.1.1. Principios

Los postulados descriptos anteriormente, inspiraron los doce principios del Manifiesto Ágil, que definen las características que diferencian un proceso ágil de uno tradicional. Dos de ellos hacen referencias a generalidades, luego hay seis que tienen que ver directamente con el proceso de desarrollo de software a seguir y cuatro que están relacionados con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo.

1) la prioridad es satisfacer al cliente mediante entregas de software tempranas y continuas;

2) los cambios en los requerimientos son aceptados y validados con los clientes;

3) software funcional se entrega frecuentemente, con el menor intervalo posible entre entregas;

4) el cliente y los desarrolladores deben trabajar juntos a lo largo del proyecto;

5) el proyecto se centra en individuos motivados;

6) el dialogo cara a cara es el método más eficiente y efectivo para comunicar información dentro del equipo;

7) el software funcional es la medida principal del progreso;

8) los procesos ágiles promueven el desarrollo sostenido;

9) la atención continua a la excelencia técnica y al buen diseño mejora la agilidad;

10) la simplicidad es esencial;

11) las mejores arquitecturas, requerimientos y diseños surgen de equipos auto organizados,

12) el equipo reflexiona en cómo ser más efectivos, y ajusta su comportamiento en consecuencia.

Aunque los creadores e impulsores de las metodologías ágiles más populares han suscrito el manifiesto ágil y coinciden con los principios

enunciados anteriormente, cada metodología tiene características propias y hace hincapié en algunos aspectos más específicos.

Los métodos ágiles más populares actualmente son:

- Extreme Programming (XP)
- Scrum
- Adaptive Software Development (ASD)
- Crystal Clear y otras metodologías de la familia Crystal
- Feature Driven Development
- Lean software development

A continuación se describe las características principales de Scrum, para entender el proceso MockupDD que se basa en esta metodología, y esta es a su vez la base de la investigación.

2.2.1.2. Scrum

Scrum es un marco de referencia para desarrollo ágil de productos de software. Inicia con la premisa de que el mundo es complicado y por lo tanto "*no se puede predecir o planear definitivamente lo que se entregará, cuando se entregará y que calidad y costo tendrá*" (Ken Schwaber). No se basa en seguir un plan sino en la adaptación continua a las circunstancias de la evolución del proyecto. Mientras que otras metodologías como XP propone ciertos estilos o técnicas de programación (programación de a pares, estándares de codificación, refactorización), Scrum hace énfasis en la administración de proyecto.

Scrum es un proceso que aplica de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos [25].

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por la funcionalidad real que aportan al interesado en el desarrollo del proyecto. Por ello, Scrum está especialmente indicado para proyectos en

entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

2.2.1.2.1. Principales elementos

Existen dos aspectos fundamentales a diferenciar en Scrum que son: los actores y las acciones; los actores son los que ejecutan las acciones. Los actores contemplados en esta metodología son:

Product Owner: conoce y marca las prioridades del proyecto o producto.

Scrum Master: es la persona que asegura el seguimiento de la metodología guiando las reuniones y ayudando al equipo ante cualquier problema que pueda aparecer. Ayuda al equipo a focalizarse en el trabajo que se debe desarrollar para cumplir los objetivos.

Scrum Team: son las personas responsables de implementar la funcionalidad o funcionalidades elegidas por el Product Owner.

Usuarios o Cliente: son los beneficiarios finales del producto, y son quienes viendo los progresos, pueden aportar ideas, sugerencias o necesidades.

Las acciones de Scrum forman parte de un ciclo iterativo repetitivo, por lo que el mecanismo y forma de trabajar que a continuación se indica, tiene como objetivo minimizar el esfuerzo y maximizar el rendimiento en el desarrollo.

Product Backlog: son todas las tareas, funcionalidades o requerimientos a realizar que son marcadas por el Product Owner, se hace uso Historias de Usuario que son un enfoque de requerimientos ágil y se focalizan en establecer conversaciones acerca de las necesidades de los clientes, son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea dicha funcionalidad.

Sprint Backlog: una o más tareas que provienen del Product Backlog y deben ser realizadas en unas 2 o 4 semanas. Existe una norma fundamental que mientras un Sprint Backlog se inicia no debe ser alterado o modificado, hay que esperar a que concluya para hacerlo.

Dayli Scrum Meeting: es una tarea iterativa que se realiza todos los días que dure el Sprint Backlog con el equipo de desarrollo, su objetivo es identificar obstáculos o riesgos que impidan el normal avance, verificar el avance de las tareas y la planificaciones de las mimas para el día.

Sprint Planning Meeting: son reuniones cuyo objetivo es planificar el Sprint Backlog a partir del Product Backlog, suelen participar el Scrum master, Scrum Team y el Product Owner.

Sprint Review: una vez finalizado un Sprint Backlog, se revisa en aproximadamente dos horas si se ha obtenido un producto que pueda ser evaluado por el cliente o usuario, un Scrum Team es quien muestra los avances.

Sprint Retrospective: el Product Owner revisará con el equipo los objetivos marcados inicialmente en el Sprint Backlog concluido, se aplicarán los cambios y ajustes si son necesarios, y se marcarán los aspectos positivos (para repetirlos) y los aspectos negativos (para evitar que se repitan) del Sprint Backlog.

2.2.1.2.2. Proceso

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos. Cada iteración proporciona un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

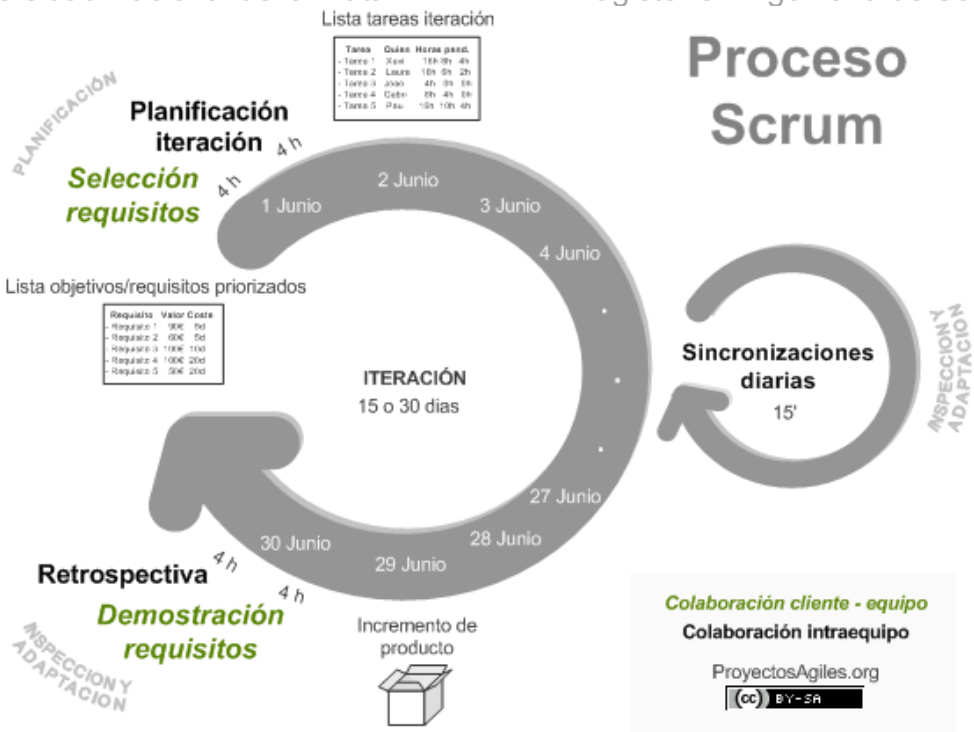


Figura 2. Proceso Scrum

El proceso parte del Product Backlog, que actúa como un plan del proyecto, de esta lista el cliente prioriza los requisitos basándose en objetivos, balanceando el valor que le aportan y su coste, quedando repartidos en iteraciones y entregas (Sprint Backlog). De manera regular el cliente puede maximizar la utilidad de lo que se desarrolla mediante la re planificación de objetivos que se puede realizar al inicio de cada iteración.

Cada día de una iteración debe realizarse una Dayli Scrum Meeting con Scrum Team con el objetivo de obtener de primera mano los avances de las tareas y los obstáculos que se van presentando a lo largo del desarrollo de la iteración

Una vez finalizado un Sprint Backlog, se revisan con el usuario o cliente los productos obtenidos (Sprint Review) y si cumplen con las necesidades plasmadas por el usuario al inicio de la iteración. Cada fin de un Sprint Backlog, se debe revisar los aspectos positivos y negativos del mismo con el objetivo de

poder utilizar estos para una mejor planificación de la siguiente iteración a realizar.

2.2.2. Proceso de Desarrollo Ágil para la Web

El desarrollo de aplicaciones web tiene un tratamiento distinto al de una aplicación tradicional. Como los requerimientos muchas veces son desconocidos o bien variables los procesos ágiles pueden acomodarse a este tipo de desarrollo, ya que el supuesto de predictibilidad del conjunto de requerimientos con que trabajan los enfoques tradicionales no se cumplen.

Al usar un método ágil de desarrollo, por la flexibilidad que estos tienen, se explica al cliente de que no existe una forma extremadamente planificada de hacer las cosas, lo cual puede ser para algunos algo chocante, la ventaja de esta menor planificación del esquema ágil sobre un método tradicional es la flexibilidad a los cambios de requerimientos [26].

Una aplicación web, cumple en gran parte las características indicadas para el tipo de proyecto donde utilizar métodos ágiles puede resultar beneficioso. Las necesidades de contar con este tipo de aplicaciones son por lo general contra el tiempo, y con requerimientos inestables. Un cliente que contrata un proyecto web requiere que su producto esté disponible en la red lo más pronto posible. Si no se contempla esto, la aplicación no resultará un producto satisfactorio para el cliente. Como los procesos ágiles permiten obtener versiones de producto previas a la versión final, si se aplican adecuadamente estos procesos los clientes podrán disponer de forma rápida de alguna versión intermedia. Además el ciclo de desarrollo de la mayoría de los sitios y aplicaciones Web es extremadamente corto. Esto implica que generalmente no se aplique ningún tipo de proceso.

2.3. MockupDD: Modelos + Agilidad

MockupDD[27] es un enfoque basado en modelos que combina prácticas Model-Driven tradicionales con características de enfoques de desarrollo ágil. El proceso comienza con la construcción de mockups para la recolección de

requerimientos y son utilizados a lo largo del proceso, así, los prototipos son enriquecidos para la posterior generación de modelos MDWE.

2.3.1. Beneficios del uso mockups

Los mockups son prototipos que muestran cómo la interfaz de usuario de un sistema de software supone que se vea durante su interacción con el usuario final. Pueden ser muy simples, sólo para ayudar a la presentación de la interacciones usuario-sistema, o más detalladas.[28]

Su principal objetivo es ayudar en la discusión de las especificaciones de interfaz de usuario con los usuarios finales, así como descubrir y definir requerimientos de interfaz de usuario en un lenguaje familiar para ellos, a diferencia de las especificaciones de texto sin formato[29].

Un estudio[30] ha determinado que el uso de mockups disminuye errores en la captura de requerimientos. Un efecto secundario del uso de estos prototipos radica en que diversas ambigüedades e incluso errores se descubrieran durante la redacción de la especificación de requerimientos. El estudio también determinó que el uso de mockups tuvo un gran efecto estadísticamente significativo tanto en la comprensión de los requerimientos funcionales eficacia (mejora + 69% de media) y la eficiencia para llevar a cabo tareas de comprensión (+ 89% de mejora en promedio).

La adopción de mockups también considera los costes de su creación en el proceso de ingeniería de requerimientos. Es crucial saber si el costo adicional necesario para desarrollarlos se equipara con la mejora de la comprensión de las necesidades funcionales. En esa dirección, se realizó una investigación [31] que reunió información sobre el tiempo para definir y diseñar mockups con la herramienta Pencil [32] y luego se recogieron las percepciones de los participantes sobre el esfuerzo por construir mockups . El principal hallazgo del estudio indicó que los participantes percibieron la construcción de los prototipos como una actividad no costosa que requiere un bajo esfuerzo en comparación con el necesario para desarrollar Casos de Uso.

Hay dos tipos de mockups: desechables y los evolucionistas. Estos últimos son el núcleo de toda la metodología ágil. El primero podría ser utilizado para apoyar la comunicación de los clientes con los desarrolladores sin embargo, su desarrollo tenían que ser muy barato y muy rápido

En la actualidad existen varias herramientas que facilitan la creación de mockups, a continuación se presenta una lista de las más utilizadas

Tool	URL
Balsamiq Mockups	www.balsamiq.com/products/mockups
Pencil Project	pencil.evolus.vn/
OverSite	taubler.com/oversite/
GUI Design Studio	www.carettasoftware.com/guidesignstudio/
Axure	www.axure.com/
Moqups	moqups.com
MockFlow	www.mockflow.com/
Mockingbird	gomockingbird.com/
SketchFlow	www.microsoft.com/expression/products/ Sketchflow_Overview.aspx
iPlotz	iplotz.com

Figura 3. Herramientas para crear mockups.

2.3.2. Proceso

MockupDD inicia con una breve recolección de requerimientos, que se traduce en un conjunto de Historias de Usuario y mockups (Figura 4, paso 1), pueden ser creados con herramientas digitales y llevadas a formato HTML o directamente ser creados en formato HTML. El equipo de desarrollo captura los principales conceptos en los mockups construidos en un modelo *Structural User Interface* (*SUI*) y preserva la correspondencia entre ambos (Figura 4, paso 2). Desarrolladores y stakeholders etiquetan los mockups con anotaciones que representan la semántica de las Historias de Usuario y se obtiene un modelo SUI completamente enriquecido (Figura 4, paso 3). En este punto se puede correr una demo y validar los requerimientos (Figura 4, paso 4.a), este proceso se puede aplicar de manera incremental hasta refinar los requerimientos tanto como

sea necesario. La iteración termina con los modelos generados y la aplicación final.

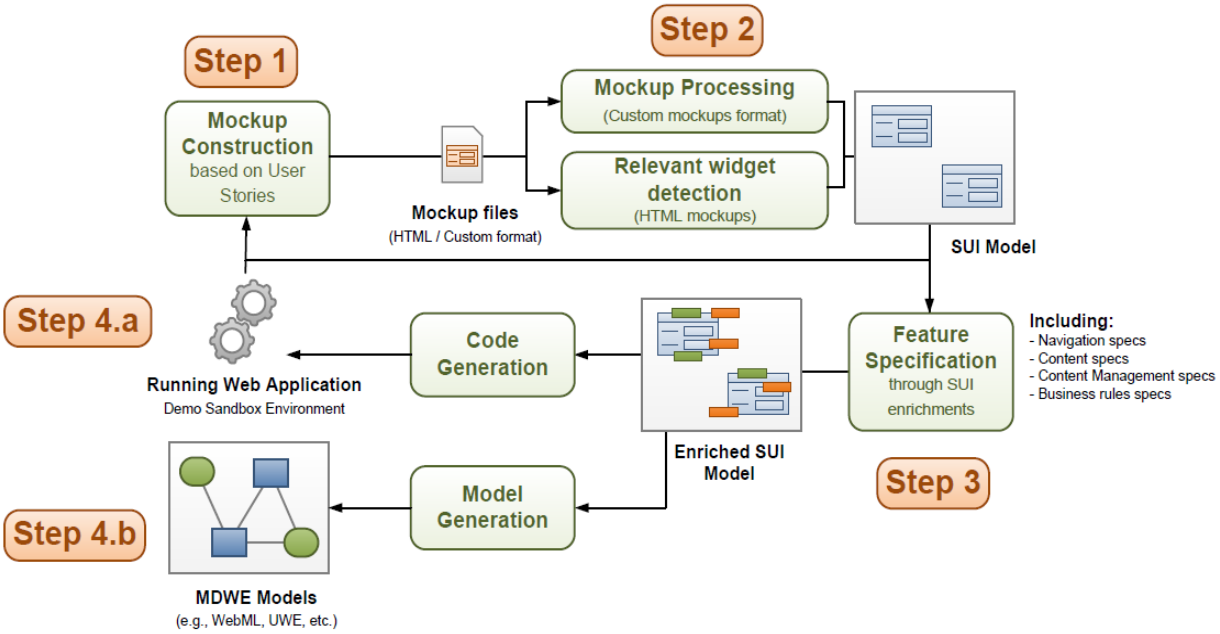


Figura 4. Flujo de una iteración de MockupDD.

- **Paso 1 y 2. Construcción de Mockups y procesamiento**

MockupDD inicia con la construcción de mockups necesarios para cada iteración, el número de interfaces de usuario depende de cómo se distribuyen los pasos de interacción entre los nodos web, una o varias Historias de Usuario son representadas mediante un prototipo.

Para formalizar la estructura de interfaz de usuario, se utiliza el SUI meta-modelo[33] su estructura se puede observar en la Figura 5 - a. El SUI meta-modelo en MockupDD define una clase abstracta Widge, pueden ser SimpleWidgets (atómicas) o CompositeWidgets (widgets de contenedores), y están agrupados en unidades de navegación (Páginas). Sin embargo, en lugar de ser sólo la definición de una estructura de interfaz de usuario, el SUI meta-modelo está diseñado para soportar la especificación de un conjunto ampliable y personalizable de características que utilizan elementos de interfaz de usuario a través de las tags, que se explicará en el siguiente apartado.

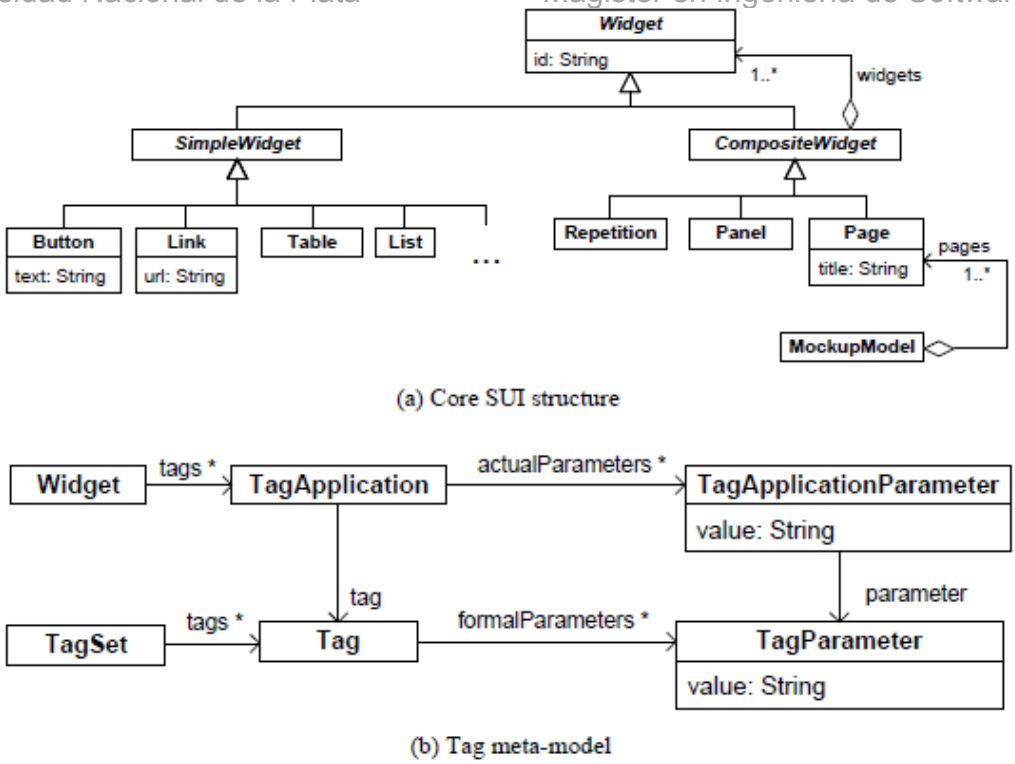


Figura 5. SUI Meta-modelo

Paso 3. Especificación de características

Mockups (en este paso SUI meta-modelo) se enriquecen con diferentes tipos de especificaciones utilizando el concepto de tag. Un tag es un enriquecimiento atómico compuesto por un nombre y cero o más los parámetros (por ejemplo, Tag (Param1, Param2,...paramN). Cada tipo de tag se puede aplicar sólo sobre un subconjunto de widgets SUI, define una sintaxis a medida para cada parámetro, extendiendo su semántica tanto como sea necesario.[34]

Los tags se agrupan en conjuntos (llamados tag sets) con el fin de aislar características específicas como navegación o manejo de datos que pueden abordarse por separado en la mayoría de los casos. A través de esta separación, se evita las dependencias secuenciales entre los enfoques especificados durante el modelado tradicional MDWE que retrasa la prueba directa del producto por los clientes o usuarios finales. La Figura 6 describe los tags

incluidos en la especificación MockupDD, junto con su semántica, aplicabilidad y el conjunto al que pertenecen.

Tag set	Tag structure and semantics	Applicable over
Navigation/Interaction	**Node(<nodeId>)** "This page will be indentified with the id <nodeId>"	Page
Navigation/Interaction	**Link(<nodeId>)** "Clicking this Link/Button will trigger a navigation to previously identified <nodeId> Page"	Button/Link
Content	**Data(<typeName>)** "This widget must show or allow editing a(n) <typeName>"	Widget
Navigation/Interaction	**Select()** "This widget will be used to mark objects in order to perform some action"	SimpleWidget
Navigation/Interaction	**Transfer(<type1, ..., typeN)** "When navigating, <type1>, ... and <typeN> will be transferred to the destination page"	Button/Link
Content	**Save([<type1, ..., typeN>])** "Clicking that Link/Button will save <type1>, ... <typeN>"	Button/Link
Content	**Delete([<type1>, ..., <typeN>])** "Clicking that Link/Button will delete <type1>, ... <typeN>"	Button/Link
Content	**Associate(<type1>, <type2>[, <associationName>])** "Clicking that Link/Button will specify that <type1>'s <associationName> will be <type2>"	Button/Link
Content	**Dissociate(<type1>, <type2>[, <associationName>])** "Clicking that Link/Button will specify that <type1>'s <associationName> will not be <type2> anymore"	Button/Link
Content	**Query(<description>, <type>)** "Widgets content in this panel will be used to get a list of <type> through the query expression <description>"	Panel
Content	**Action(<action> [, <type1>, ..., <typeN>])** "Clicking in that Link/Button will trigger a(n) <action> action, involving <type1>, ..., <typeN>"	Button/Link

Figura 6. Principales tags de MockupDD

La Figura 7 representa como un mockup que representa la Historia de Usuario Registrar Carpetas es etiquetado para describir el comportamiento que se espera.

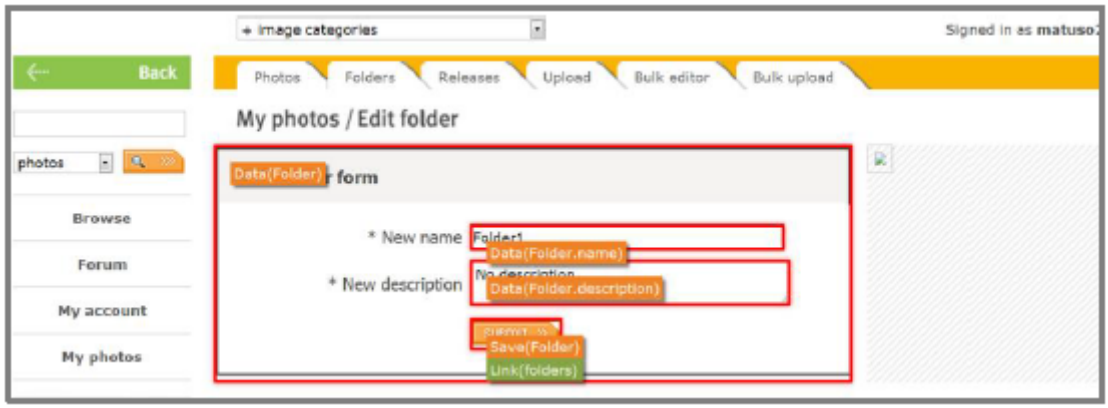


Figura 7. Mockup etiquetado. MockupDD

- **Paso4. Generación de código y modelo**

Los tags son traducidos a elementos MDWE y se combinan para especificar características de diseño más complejas. Cada iteración del proceso MockupDD implica agregar, cambiar o eliminar tags. Una vez finalizada la definición de los tags, se utilizan generadores de modelos para obtener los modelos MDWE, se cuenta con la implementación de generadores para UWE y WebML.

2.3.3. Proceso de desarrollo

El proceso de desarrollo de MockupDD es una adaptación a Scrum [1], el Sprint Backlog son mockups asociados a las Historias de Usuario. A diferencia de Scrum que continúa con codificación directa, una iteración utiliza interfaces de usuario creadas para concretar requerimientos, los mockups son traducidos a representaciones SUI y se etiquetan con la herramienta de soporte provista por la metodología. Al completar el etiquetado, puede obtenerse una demo de modo que usuarios finales y el Product Owner pueden evaluar la aplicación y validar que se comporte como ellos lo esperaban, sin esperar al final del Sprint [34].

2.3.4. Herramientas de soporte

MockupDD se apoya en las siguientes herramientas:

Mockup-to-HTML: genera una representación HTML de mockups que no se encuentran en este formato.

Herramienta interactiva de etiquetado: permite crear un modelo SUI sobre una estructura HTML. Después de que el HTML y el modelo se completan y son correctamente mapeados, es posible aplicar los tags. Para ello, la herramienta puede funcionar en dos modos diferentes: Widget de descubrimiento (para construir el SUI) y widget de etiquetado (para aplicar tags sobre los widgets mapeados).

Demo Sandbox Environment: Una vez que los tags son suficientes para satisfacer los requerimientos de una iteración, esta herramienta permite generar rápidamente una versión que puede ser evaluada por los usuarios finales. La herramienta no depende de la metodología MDWE subyacente; tampoco requiere ningún despliegue de aplicaciones para emular la aplicación final en ejecución. Esta herramienta utiliza *sandbox* para enriquecer los archivos HTML estáticos (generados a partir de los Mockups o bien partiendo de los HTML importados) con el comportamiento expresado por los tags.

La herramienta de etiquetado Interactiva y Demo Sandbox environment realizan las tareas más importantes.

14.1 CAPÍTULO 3. ESTRATEGIA PROPUESTA

3.1. Introducción

En el capítulo anterior se realizó una revisión de las metodologías ágiles y las dirigidas por modelos, al rescatar lo mejor de las dos y en busca de generar una alternativa, surge MockupDD que propone un enfoque ágil para MDWE, se apoya en la construcción de mockups que facilitan la generación de prototipos y modelos de software, como resultado, se obtiene un proceso iterativo ágil con ventajas de uno MDWE. Sin embargo, tratar conceptos del lenguaje para implementar requerimientos específicos sigue presentado las mismas dificultades que en los procesos MDWE convencionales: es necesario agregar nuevos elementos al metamodelo (en este caso representado por tags) y modificar los generadores e intérpretes de código.

El objetivo del presente trabajo es proponer una mejora a MockupDD, en donde implementar requerimientos específicos no contemplados en los elementos del lenguaje, no presente dificultades ni requiera de conocimientos avanzados.

La metodología al igual que MockupDD propone partir de mockups, y se considera que los mismos son suficientes para inferir la estructura dinámica de la aplicación. Como se comentó en el capítulo anterior, el uso de mockups trae consigo beneficios en la captura de requerimientos que consiste en detectar errores en las primeras etapas de la construcción del software. Además se ha demostrado que el tiempo que toma la construcción de estos prototipos se equipara con la mejora en la comprensión de las necesidades funcionales. MockupDD es posible gracias al desarrollo de un lenguaje basado en tags, que permite etiquetar un mockup con información suficiente para que las tareas y flujos de trabajo de desarrollo web comunes puedan ser eliminadas o simplificadas.

El beneficio del enfoque planteará una arquitectura de tags que permita de manera sencilla ajustar requerimientos que excedan las capacidades del lenguaje. La metodología propuesta EMockupDD (Extensible MockupDD) se

detalla a continuación y en el siguiente capítulo se implementará MetapiTag herramienta que permitirá llevar a cabo el proceso descrito.

3.2. Tipos de proyectos

EMockupDD es aplicable a proyectos que apunten a sitios web, en donde:

- La mayoría de las funcionalidades del sistema, puedan ser expresadas mediante mockups, y para el resto de características que no pueden traducirse en estos prototipos, existe la posibilidad de su implementación vía codificación manual.
- La navegación entre las páginas sea clara.

3.3. Aplicación Ejemplo

Con el fin de explicar el proceso EMockupDD, se desarrollará un sitio web para la venta de libros (BookStore). Se podrá listar libros, realizar filtros sobre la lista, ver el detalle de cada libro y registrar usuarios al sitio.

A continuación se definen algunas Historias de Usuario que se consideran apropiadas para explicar la metodología.

- Historia de Usuario 1 (HU1). Listar los libros de la tienda con sus características principales.
- Historia de Usuario 2 (HU2). Ver el detalle de un libro de la lista anterior.
- Historia de Usuario 3 (HU3). Crear un usuario dentro del sitio.

3.4. Proceso de desarrollo

Figura 8. Proceso EMockupDD

EMockupDD utiliza mockups como artefactos centrales que impulsan el proceso de desarrollo, es una adaptación a Scrum[1] en donde el Sprint Backlog está representado por Historias de Usuario y mockups (Figura 8, paso 1). El equipo de desarrollo captura los principales conceptos en los mockups construidos en un SUI meta-modelo (Figura 8, paso 2). El equipo de desarrollo junto con las partes interesadas interpretan la semántica en las Historias de

Usuario y mockups, especifican los requerimientos y se etiquetan cada uno de ellos y se obtiene un modelo SUI completamente enriquecido (Figura 8, paso 3), hasta este punto no existe diferencias con la versión original de MockupDD.

Completado el enriquecimiento se obtiene una aplicación que puede ser evaluada de manera inmediata, a diferencia de MockupDD tradicional, los tags, no son traducidos en modelos, sino se convierten en código ejecutable que inyecta dinamismo a los prototipos estáticos (Figura 8, paso 4). Al ser una estrategia que parte de mockups que son la base de los procesos ágiles se plantea seguir con la adaptación a Scrum, pero no está restringida a estas metodologías.

3.4.1. Paso 1 y 2. Construcción de Mockups y procesamiento

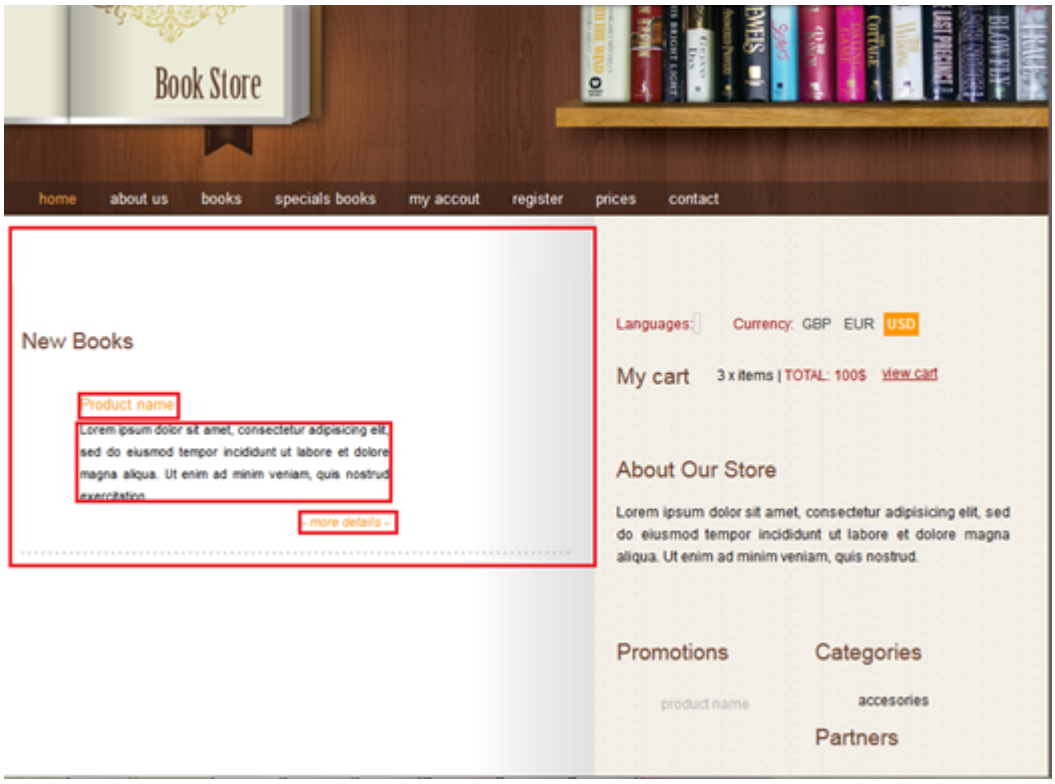
EMockupDD inicia con la construcción de mockups, el alcance de este trabajo no especifica cómo , MODFM [35] propone un algoritmo para la creación de estas interfaces de usuario, de una manera óptima y probada.

1	Descomponer funcionalmente el sistema en menús. Cada elemento del menú se identifica como una página Web.
2	Generar mockups únicamente con el sistema de menús y páginas falsas en una versión comercial.
3	Desplegar los mockups en un servidor de pruebas. Volver al paso 1 si el cliente quisiera algunos cambios; de lo contrario ir con el paso 4.
4	Iterar en cada paso del menú del 4 al 7 Si se termina ir al paso 8.
5	Desarrollar escenarios de cada página del menú. Identificar una lista de páginas, reúne información de la página, y registrar la lógica de negocio entre las páginas.
6	Generar un mockup, con la lógica de negocio que aparezca en cada página como documento escrito.
7	Entregar la mockups al cliente. Vuelva al paso 5 si cliente le gustaría algunos cambios; si no, vaya con el paso 4.
8	Entregar el mockup a los desarrolladores.

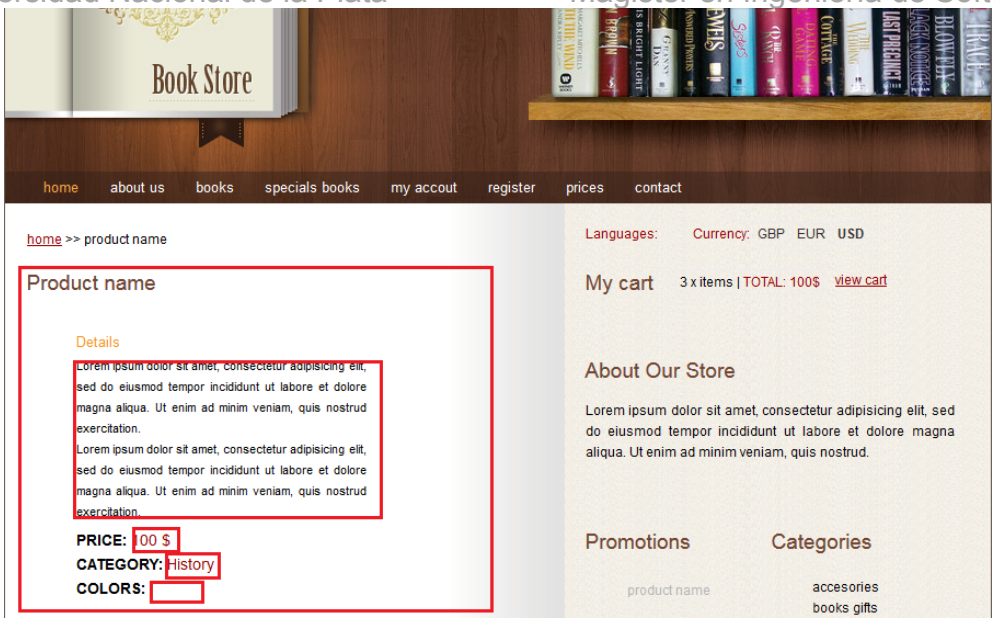
Tabla 1. Algoritmo MODFM

Al igual que la MockupDD, para formalizar la estructura de interfaz de usuario se utiliza SUI meta-modelo.

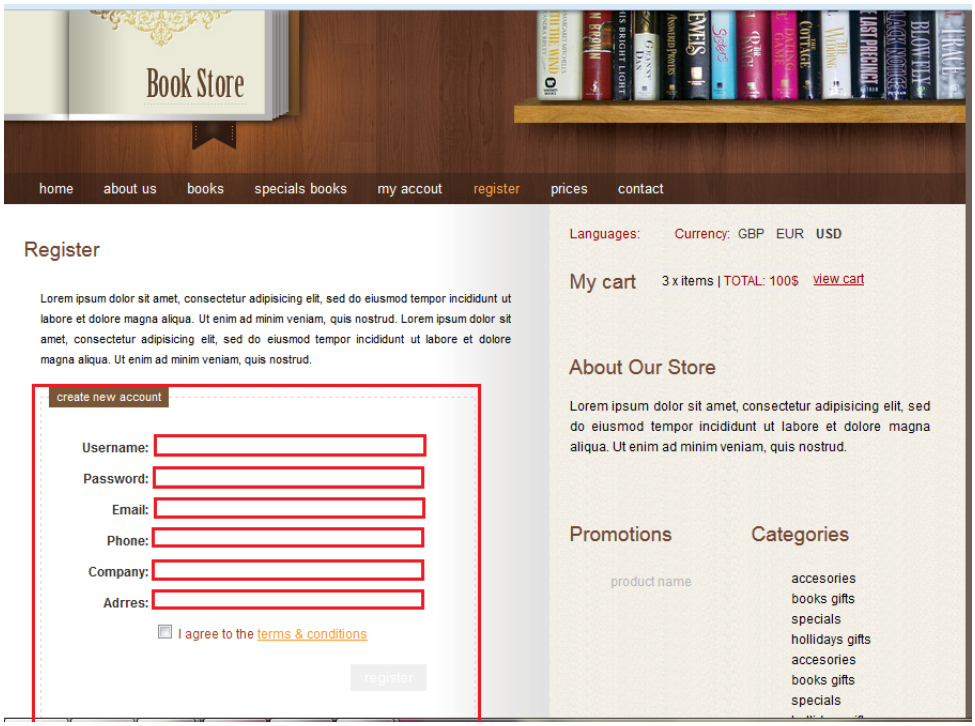
La Figura 9 (a-b-c) presenta los mockups construidos para las Historias de Usuario antes mencionadas (Aplicación Ejemplo). Una instancia del modelo SUI ha sido extraída después de su creación, los widgets que son importantes desde el punto de vista las Historias de Usuario se muestran resaltados, los cuadros de color rojo son una decoración visual añadida para denotar los widgets ya asignados a los elementos subyacentes SUI. Una vez seleccionados y asignados todos los widgets pertinentes de acuerdo con los conceptos de la Historia de Usuario, se pasa a la especificación de las características de estos widgets.



(a). Mockup HU1 – HU2. index.html



(b). Mockup HU2. detailBook.html



(c). Mockup HU3. register.html

Figura 9. Mockups BookStore.

3.4.2. Paso 3. Especificación de características.

El equipo de desarrollo junto con las partes interesadas interpretan la semántica en los mockups (en este punto SUI Meta-modelo) y especifican los requerimientos etiquetándolos con tags, como en MockupDD tradicional.

Un tag conserva su estructura original, está representado por un nombre y un conjunto de parámetros que puede ser cero o más. Estos parámetros contienen información necesaria para inyectar dinamismo a los mockups. A diferencia de MockupDD, cada tag se convertirá en código ejecutable, que puede apuntar tanto al lado servidor como al lado cliente, la Figura 10 muestra como el mockup para la HU1 fue etiquetado.

Figura 10. Mockup de la Figura 9 (a) enriquecido con tags de EMockupDD.

EMockupDD permite que un tag pueda estar compuesto por n-tags hijos, los mismos que son invocados por su tag padre, y son *dependientes* de el para funcionar, en la Figura 10 los tags que hacen referencia a los atributos de la lista `name` y `description` son tags hijos o dependientes de `ListTag`. A pesar que está posibilidad permite crear tags complejos, la lógica de negocio no puede ser cubierta con un conjunto finito de tags. El valor agregado de EMockupDD radica en que, a diferencia de MockupDD que cuenta con un conjunto de tags estático, en donde, agregar o modificar el comportamiento de los mismos presenta

dificultades ya comentadas, la estrategia es extensible, permite modificar e introducir nuevos tags, sin mayores complicaciones, ni el conocimiento de expertos y en tiempo de ejecución.

3.4.3. Conjunto de tags básico

Para demostrar que EMockupDD funciona de acuerdo a lo especificado, en el capítulo siguiente se construye MetapiTag con un conjunto de tags básicos, cada tag si bien consta de n-parámetros, estos se agrupan en <parameters> y varían según las necesidades de cada tag, se necesita el <selector> sobre el cual se realiza el enriquecimiento y <subTags> en donde se indica por que tags está formado.

3.4.3.1. ListTag

Permite mostrar una lista de elementos del dominio.

- <selector>: hace referencia al elemento padre de la lista (donde se agrega cada ítem de la lista)
- <parameters>: Lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.
  - <content>: indica de qué tipo de objetos está formada la lista.
  - <cloneElement>: indica dentro del DOM seleccionado, el elemento que se repetirá para cada objeto.
- <subTags>: lista de tags hijos que se van a aplicar sobre cada elemento de la lista. Puede contener tags del tipo PropertyListTag que permite indicar cada uno de los atributos de la lista.

ListTag obtiene una lista de objetos <content>, clona el contenido de <cloneElement> para cada objeto, e invoca cada uno de sus hijos con un objeto de la lista y el DOM creado para ese objeto. Cada tag hijo aplicará alguna acción en base a ambos. Luego, el DOM modificado se agrega al tag padre.

3.4.3.2. PropertyListTag

Permite mostrar los atributos que se visualizarán de una lista.

- <selector>: hace referencia al elemento dentro de la lista, que indica el atributo a presentar.
- <parameters>: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.
  - <content>: nombre del atributo del objeto que se muestra en la lista.
  - <selectorTag>: indica dentro del DOM seleccionado, el elemento al que se hace referencia.
- <subTags>: es un tag que no se compone de otros.

Es un tag dependiente porque necesita de ListTag para funcionar, este le envía un objeto DOM y un objeto del dominio, busca el elemento con <selectorTag> dentro del DOM recibido y le asigna el resultado de evaluar la propiedad <content> sobre el objeto particular recibido.

3.4.3.3. DeleteListTag

Permite eliminar elementos seleccionados de una lista.

- <selector>: hace referencia al elemento, sobre el cual se ejecutará la operación de borrado.
- <parameters>: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.
  - <event>: nombre del evento que se le agrega al elemento cuyo selector es <selector>.
- <subTags>: es un tag que no se compone de otros.

Es un tag dependiente, ya que recibe un objeto de `ListTag`, al elemento con <`selector`> se le agregar el evento <`event`> para que elimine el objeto que recibe del tag padre.

3.4.3.4. DataEntryTag

Permite formar un objeto del dominio para crearlo, editarlo o modificarlo, está formado por tags del tipo `DataPropertyTag` que indican las propiedades del objeto,

- <`selector`>: hace referencia al elemento que contendrá los atributos que formarán el objeto.
- <`parameters`>: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.
  - <`content`>: nombre del objeto que se quiere formar.
- <`subTags`>: hace referencia a los tags hijos, que representan las propiedades que conforman el objeto.

Es un tag contenedor que crea un objeto con atributos que son indicados por medio de sus tag hijos. Este tag también puede ser padre de `SaveTag`, `DeleteTag` o de cualquier tag que reciba un objeto para procesarlo de algún modo

3.4.3.5. DataPropertyTag

Sirve para especificar los atributos del objeto que representa `DataEntryTag`.

- <selector>: hacer referencia al elemento que representa los atributos del objeto formado por `DataEntryTag`.
- <`parameters`>: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.
  - <`content`>: nombre de cada atributo del objeto.

- <subTags>: es un tag que no se compone de otros.

Es un tag dependiente, recibe un DOM y un objeto del dominio del tag padre. Busca el elemento con <selector> dentro del DOM recibido y le asigna el resultado de evaluar la propiedad <content> sobre el objeto particular recibido.

3.4.3.6. SaveTag

Persiste un objeto que es enviado desde DataEntryTag a la base de datos.

- <selector>: hace referencia al elemento sobre el cual se ejecutará la operación de persistir.
- <parameters>: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.
  - <event> nombre del evento, que se agrega al elemento cuyo selector es <selector> para que proceda con la acción.
- <subTags>: tags hijos en caso de tenerlos.

Es un tag dependiente, ya que recibe un objeto de su tag padre. Al elemento con <selector> se le agregar el evento <event> para que persista el objeto que recibe del tag padre.

3.4.3.7. DeleteTag

Elimina un objeto que es enviado desde un tag padre que puede ser DataEntryTag.

- <selector>: hace referencia al elemento sobre el cual se ejecutará la operación de eliminado.
- <parameters>: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.

- <event>: nombre del evento que se le agrega al elemento cuyo selector es <selector>, para que elimine el objeto

- <subTags>: tags hijos en caso de que tuviera-.

Es un tag dependiente ya que recibe un objeto del tag padre para eliminarlo. Al elemento con <selector> se le agregar el evento <event> para que realice la acción de eliminar el objeto que recibe del tag padre.

3.4.3.8. LinkTag

Permite re direccionar a otra página.

- <selector>: selector CSS del elemento sobre el cual se agrega la acción para re direccionar a otra página.
- <parameters>: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.
  - <page>: nombre de la página a la que se re direcciona.
- <subTags>: es un tag que no se compone de otros.

Busca el elemento con <selector> y le agrega un manejador para que se re direccione a la página <page> frente a un evento de click.

3.4.3.9. TransferTag

Va a la par de un link y permite llevar un objeto hacia otra página.

- <selector>: hace referencia al elemento sobre el cual se agrega la acción para que un objeto proveniente de un tag sea transferido a otra página u otra sección dentro de la misma página.
- <parameters>: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.
  - <page>: nombre de la página a la que se necesita transferir un objeto.

- `<idTag>`: id del tag al que se transfiere el objeto.

- `<subTags>`: es un tag que no se compone de otros.

Es un tag dependiente ya que recibe un objeto del tag padre, y lo almacena mediante un identificador que se compone de la página destino y el identificador del tag, utiliza LocalStorageDataManager para almacenar el objeto a transferir.

3.4.3.10. DataTag

Permite recibir un objeto que es transferido mediante el tag anterior.

- `<selector>`: hace referencia al elemento que agrupa la información del objeto que es recibido.

- `<parameters>`: no necesita de parámetros adicionales.

- `<subTags>`: hace referencia a los tags hijos, que representan las propiedades que se visualizan del objeto que es recibido.

Es un tag contenedor que sirve para recibir un objeto que es enviado desde otra página. Es creado previo la creación de `TransferTag`, ya que para la configuración de este se necesita conocer identificador de `DataTag`.

3.4.3.11. DataNavTag

Permite navegar un objeto relacionado con otro proveniente de su tag padre.

- `<selector>`: hace referencia al elemento que hace se relaciona con otro objeto.

- `<parameters>`: lista de parámetros necesarios para el funcionamiento correcto del tag, a continuación se lista cada uno de ellos.

  - <`content`>: hace referencia al objeto relacionado.

  - <`key`>: es el nombre de la propiedad por la que se relaciona el objeto al que se accede.

Es un tag dependiente recibe un objeto y evalúa la propiedad <key>, este valor servirá para obtener un único elemento <content> que es el objeto relacionado. Puede contener tags que permitan leer o escribir valores de las propiedades de un objeto, otros DataNavTag que permitan navegar hacia otros objetos.

El código más significativo de cada uno de los tags explicados puede consultarse en el Anexo A.

3.4.4. Paso 4. Generación de código

EMockupDD ya no traduce los tags en elementos MDWE como lo planteado en la versión original de MockupDD. Se propone recoger el conjunto de parámetros que posee, de requerirse trabaja en conjunto con otros tags para formar bloques de comportamiento siguiendo la estructura indicada por ellos, ejecuta la lógica indica tanto del lado cliente como servidor si fuese necesario.

En caso de que un tag no cumpla con los requerimientos necesarios, se puede crear nuevos tags o en su defecto modificar directamente el comportamiento del mismo, introduciendo una gran ventaja que supera una limitación en la metodología original. Cada iteración del proceso permite agregar, eliminar, modificar tags.

3.5. Valor Agregado

El valor agregado de este trabajo de investigación sobre MockupDD, radica en:

MockupDD a partir de mockups enriquecidos, genera modelos que permiten derivar una aplicación web funcional (por ejemplo, modelos WebML o UWE). EMockupDD no genera modelos pero los tags elevan la abstracción conservando una de las ventajas de MDD la diferencia está en la implementación y en su arquitectura extensible que logra independencia y agilidad.

MockupDD está restringida a las especificaciones del lenguaje de los modelos que utilice, tratar requerimientos que excedan las capacidades expresivas del lenguaje utilizado, presenta dificultades, al momento de crear

nuevos elementos y por consiguiente se necesita modificar los generadores e intérpretes de código, la utilidad de esta investigación permite de manera fácil y sin la necesidad de conocimiento de expertos agregar nuevos conceptos de lenguaje, así como modificar los existentes.

3.6. Arquitectura

El proceso indicado en el apartado anterior, se puede lograr gracias a una estructura de tags y diseño de plataforma sobre la cual se ejecutan, que se propone a continuación.

Cada tag se convierte en fracciones de código ejecutable por sí mismos que tienen comportamiento tanto en el lado cliente como en el lado servidor. Los tags en MockupDD se transforman en modelos y luego estos modelos en código, esta arquitectura evita que al agregar nuevos elementos al lenguaje se deba modificar los generadores de código.

14.1 Ejecución de tags en la plataforma

Para transformar los tags en código ejecutable que agregue dinamismo efectivo a los archivos HTML, es necesario que cada uno de ellos realice acciones en el lado cliente y en lado servidor si fuese necesario. Por ejemplo para la HU1 (Figura 10, a) el mockup es enriquecido con `ListTag` sobre la lista de libros y `PropertyListTag` sobre cada atributo de la lista, estos tag son dependientes de `ListTag`. A continuación se abordan aspectos de cómo estos tags son procesados hasta llegar a obtener una aplicación ejecutable.

14.2 Lado Cliente

El comportamiento del lado cliente estará implementado por una clase concreta `<tagName>` en JavaScript, que se podrá instanciar con un tradicional `new <tagName>()`.

Una vez que los mockups sean etiquetados y guardados, se crea la lógica del lado servidor. Para que los tags puedan ser ejecutados, se instancia cada uno de ellos y este código se incluye en cada interfaz. La instanciación de los tags para un mockup primero obtendrá los tags que le pertenecen y formará un script que los instanciará. En este punto, para determinar el código concreto que

instanciará cada tag necesita obtener el selector del elemento sobre el que se aplicó el tag, los parámetros necesarios y los subTags de los que se compone. Cuando se guardan los tags para un mockup se crea el lado servidor para el funcionamiento de la aplicación, este proceso se detalla a continuación.

14.3 Lado Servidor

El lado servidor de cada tag debe especificarse para que la lógica pueda ser creada al momento de guardar los tags para un mockup, no todos los tags necesitarán que se realice acciones en el lado servidor. Por ejemplo `ListTag` necesita recuperar los libros de la base de datos, en cambio `PropertyListTag` que hace referencia a los atributos de la lista solamente deberá evaluar esa propiedad para el objeto enviado de la lista. Básicamente la lógica de esta sección se centra en las cuatro operaciones básicas CRUD (Create, Read, Update, Delete), se usará una Base de Datos dinámica que crea objetos bajo demanda.

14.4 Guardar tags para un mockup

El modelo de tags estará almacenado en una base de datos, se considera que es necesario guardar la siguiente información:

- mockupId: identificador del mockup
- tags: lista de tags de un mockup, cada tag contendrá los siguientes atributos.
  - Selector: sobre el cual se aplica el enriquecimiento.
  - Parameters: los parámetros propios del tag
  - nameTag: Nombre del tag
  - id: identificador único del tag
  - parent: referencia al tag padre.
  - subTags: referencia a los tag hijos de ser el caso.

15.1__

16.1 CAPÍTULO 4. PROTOTIPO

4.1. Introducción

En esta sección se describen detalles de la implementación de MetapiTag herramienta que apoya EMockupDD, posteriormente se pone en manifiesto el uso de la misma, desarrollando la Aplicación Ejemplo descrita en la sección Aplicación Ejemplo, que permitirá demostrar el flujo de la metodología y la manera como MetapiTag apoya su proceso.

MetapiTag está basada en ELECTRA[36] que es un enfoque híbrido que combina principios Ágiles, MDD y permite codificación, es una versión mejorada de MockAPI[37], usa un lenguaje amigable para definir requerimientos al anotar los mockups con tags, permite generar de forma rápida una aplicación que puede ser testeada y extendida mediante codificación directa sin romper la abstracción que proporcionan los modelos MDD.

La intención del desarrollo de esta herramienta es probar EMockupDD, implementar los tags básicos planteados, y demostrar que es factible agregar y modificar tags de manera sencilla.

4.2. Visión del Prototipo

16.1 Alcance

El prototipo está diseñado e implementado de forma modular de la siguiente manera.

Módulo 1. Administración de Tags

Permite el manejo de tags, se puede manipular la lógica del lado cliente, lado servidor si fuere necesario y la información que cada tag necesita para ser etiquetado e instanciado.

Módulo 2. Administración de proyectos y sus recursos

Permite subir un proyecto mediante un archivo comprimido, los cuales contienen los mockups HTML, estilos, imágenes, scripts y lo necesario para el funcionamiento de la aplicación web.

De cada proyecto se lista los mockups y se presenta la opción para etiquetarlos y ejecutarlos.

Módulo 3. Etiquetado y ejecución de mockups

Luego de crear un proyecto, se permite etiquetar un mockup con los tags disponibles, ofrece un entorno interactivo en el cual se detectan los principales elementos del DOM y se visualiza la lista de tags disponibles. Para crear un tag se solicitará la información necesaria para su instanciación, que fue configurada en el módulo de Administración de Tags.

Luego de enriquecer los archivos HTML con los tags necesarios, inmediatamente se tiene una aplicación corriendo y puede ser ejecutada para probar su funcionamiento.

4.3. Metodología

Para el desarrollo del prototipo, se plantearon reuniones periódicas, en donde se realizaba una revisión de los avances y se planteaba nuevos hitos a cumplir siguiendo Scrum, con reuniones semanales, Springs y un Spring Backlog a cumplir.

4.4. Herramientas utilizadas

- Eclipse Java EE IDE for Web Developers.
- Metapi: permite la definición, creación y configuración de recursos en tiempo de ejecución, el acceso a ellos es por medio de una interfaz RESTful. Está basada en MockAPI[37] y ELECTRA[36], se estructura en módulos, que se encargan de tareas específicas. A continuación se mencionan las que son relevantes para la implementación a desarrollar.

EndPoint Editor: permite crear, modificar, eliminar endPoints en tiempo de ejecución siguiendo las mejores prácticas para servicios RESTful[38] y pueden ser accedidos mediante una única URL a través de operaciones HTTP convencionales:

  - GET /[recurso]
  - POST /[recurso]

- GET /[recurso]/{id}
- PUT /[recurso]/{id}
- DELETE /[recurso]/{id}

Resource Editor: permite persistir scripts, código HTML, en general cualquier recurso binario que se requiera. Luego se puede acceder a ellos. Simula el manejo de archivos y carpetas, se pueden obtener los recursos mediante los siguientes comandos.

- `getResource (pathResource)`: devuelve el recurso del PATH especificado.
- `getMergedResource(pathResource)`: devuelve todos los recursos que se encuentran bajo el PATH especificado, separados por punto y coma.

MetaBase: permite persistir objetos, usa una Base de Datos no estructurada llamada MetaBase, guarda y relaciona objetos representados como texto (lo cual da libre albedrío a su estructura y datos internos). Toda la información de los objetos será guardada en JSON. A continuación se presenta la interfaz de la herramienta:

- `getAll(String type)`: retorna todos los objetos de clase `type`.
- `getById(String type, long id)`: retorna el objeto de clase `type` cuyo id es `id`.
- `createObject(String type, String representation)`: crea un nuevo objeto de clase `type` y guarda su representación.
- `updateObject(String type, long id, String representation)`: actualiza el objeto de clase `type` con id `id`, asignándole una nueva representación.
- `deleteObject(String type, long id`: borra el objeto de clase `type` con id `id`.

- `deleteAll(String type)`: borra todos los objetos de clase `type`.

4.5. Diseño

4.5.1. Diagrama de Clases

4.5.2. Diccionario de Clases

Las clases que se describen a continuación son el motor de MetapiTag permite representar los tags, persistirlos, eliminarlos y crear un modelo de tags para un mockup.

- Tool: Clase principal de la herramienta que recupera los tags almacenados de un mockup y los asigna a Model que es la clase contenedora de tags, si el mockup no tiene tags, se crea una clase Model vacía.
- ModelSerializer: permite serializar los tags a un String para ser almacenados, y de serializarlos en forma de objetos para trabajar dentro de la herramienta.
- Model: contiene la lista de tags de un mockup. Tiene como métodos addTags() y removeTags() que hacen referencia a ModelStore.
- Tag: representa cada elemento del lenguaje.
- MetapiTag: la clase padre de la que heredan los tags implementados, contiene atributos y métodos comunes a todos los tags.
- ModelStore: contiene los métodos para persistir y obtener tags de un mockup. Se encarga también de listar los tags disponibles.

A continuación se describen las clases que hacen posible el proceso de etiquetado de tags así como su edición.

- ToolBox: clase que permite visualizar el menú desplegable al pulsar sobre los elementos principales del DOM.
- ToolBoxItem: instancia para cada elemento del menú que se presenta en la clase anterior.
- Editor: crea un nuevo TagEditor.
- TagEditor: Representa la clase que permite crear un tag, se presentan los atributos configurados para cada tag, los mismos que tendrán que ser ingresados por los usuarios.

4.6. Implementación

MetapiTag se monta como una serie de recursos, scripts y endPoints sobre Metapi, luego se crean otros endPoints para el funcionamiento del lado servidor de los tags de forma dinámica. A continuación se describen detalles de la implementación que se contemplaron a lo largo de la construcción de los módulos que forman parte de la herramienta.

4.6.1. Tag - Lado Cliente

El lado cliente de cada tag se almacena como recurso dentro de Metapi y puede accederse mediante el URL `/tags/client/<tagName>.js`. Las clases que representan los tags disponibles, se obtienen mediante el URL http://<host>:<port>/tags/client que queda expuesto vía HTTP REST a través del método GET (Figura 11), el endPoint permite importar la lógica de los todos los tags disponibles que se pueden utilizar para enriquecer los mockups.

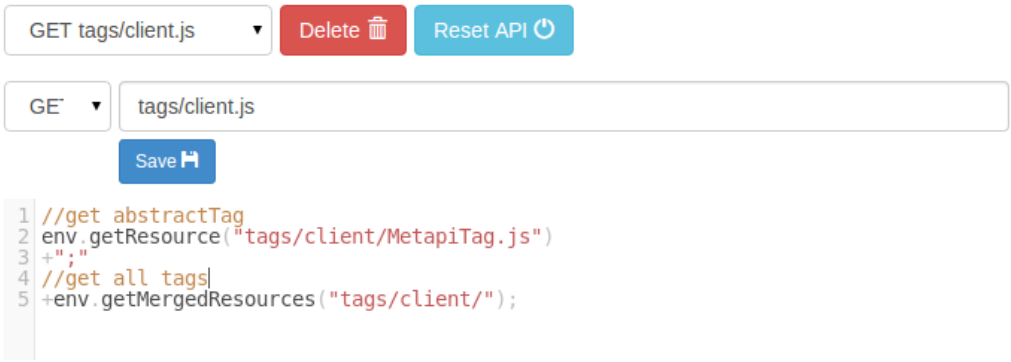
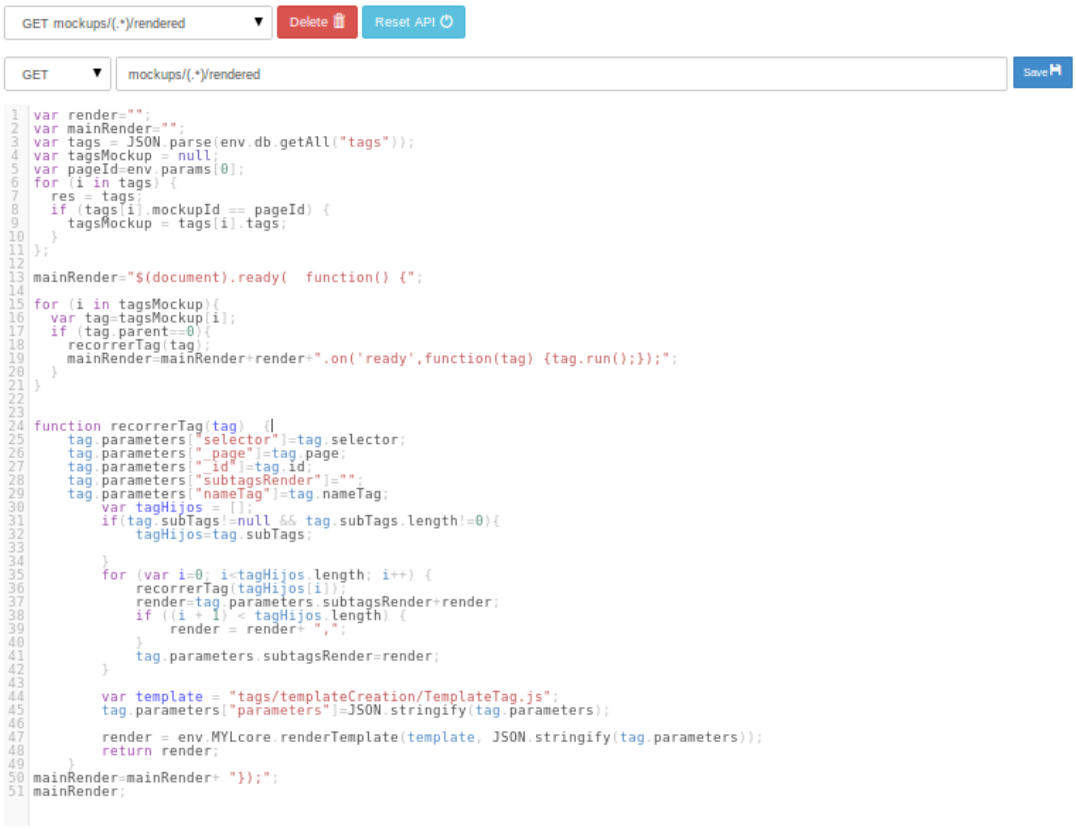


Figura 11. EndPoint que obtiene la lógica del lado cliente de los tags disponibles.

Una vez que un mockup ha sido etiquetado y se han guardado sus tags, para que el archivo HTML cobre dinamismo, se incluirá todo el código que instancia los tags, se obtiene el selector del elemento sobre el que se aplicó el tag, los parámetros necesarios para su correcto funcionamiento que se explicaron en la sección Conjunto de tags básico y los subTags de los que se compone, por medio del siguiente template `new {{nameTag}}({{&selector}}, {{&parameters}},[{{&subTags}}])` se instancia cada tag, la plantilla es general para todos los tags, lo que varían son sus parámetros, está escrita con la sintaxis

de Mustache[39], y se encuentra almacenada como recurso en /tag/templateCreation/TemplateTag.js

El JavaScript concreto que instancia los tags de un mockup, se obtiene mediante el URL http://<host>:<port>/mockups/{id}/tags/rendered que queda expuesto vía HTTP REST a través del método GET (Figura 12).




```
1  var render="";
2  var mainRender="";
3  var tags = JSON.parse(env.db.getAll("tags"));
4  var tagsMockup = null;
5  var pageId=env.params[0];
6  for (i in tags) {
7    res = tags;
8    if (tags[i].mockupId == pageId) {
9      tagsMockup = tags[i].tags;
10   }
11 };
12
13 mainRender="$(document).ready(  function() {";
14
15 for (i in tagsMockup){
16   var tag=tagsMockup[i];
17   if (tag.parent==0){
18     recorrerTag(tag);
19     mainRender=mainRender+render+".on('ready',function(tag) {tag.run();});";
20   }
21 }
22
23
24 function recorrerTag(tag)  {
25     tag.parameters["selector"]=tag.selector;
26     tag.parameters["_page"]=tag.page;
27     tag.parameters["_id"]=tag.id;
28     tag.parameters["subtagsRender"]="";
29     tag.parameters["nameTag"]=tag.nameTag;
30         var tagHijos = [];
31         if(tag.subTags!=null && tag.subTags.length!=0){
32             tagHijos=tag.subTags;
33
34         }
35         for (var i=0; i<tagHijos.length; i++) {
36             recorrerTag(tagHijos[i]);
37             render=tag.parameters.subtagsRender+render;
38             if ((i + 1) < tagHijos.length) {
39                 render = render+ ",";
40             }
41             tag.parameters.subtagsRender=render;
42         }
43
44         var template = "tags/templateCreation/TemplateTag.js";
45         tag.parameters["parameters"]=JSON.stringify(tag.parameters);
46
47         render = env.MYLcore.renderTemplate(template, JSON.stringify(tag.parameters));
48         return render;
49     }
50 mainRender=mainRender+ "});";
51 mainRender;
```

Figura 12. Código que instancia los tags de un mockup.

En resumen este código obtiene los tags aplicados para un mockup HTML (Figura 12, líneas 3-11), luego se forma el script que instancia cada uno ellos por medio del template indicado anteriormente (Figura 12, línea 44) al cual se le envía un objeto JSON que contiene la información de cada tag y, por medio de Mustache[39] se renderiza el código que instanciará cada tag . La instanciación de los tags que están compuestos por otros, se realiza por medio de una función recursiva (Figura 12, línea 24), que crea primero los tags hijos, y estos se van agregando al tag padre.

4.6.2. Tag - Lado Servidor

El lado servidor de cada tag se almacena como recurso dentro de Metapi y puede referenciarse mediante el URL `tags/server/<tagName>.js`

El acceso a los datos almacenados en la Base de Datos será mediante una URL única que queda expuesta vía HTTP REST utilizando los métodos POST, GET, PUT y DELETE para las cuatro operaciones básicas CRUD (Create, Read, Update, Delete).

Al guardar los tags para un mockup, para cada uno de ellos se instancian los endPoints necesarios por medio de un template Mustache[39] con la estructura de un objeto JSON[40] que presenta la siguiente información.

```
{"httpMethod":"{{content}}","pattern":"{{pattern}}","script":"{{scritp}}
"}
```

`httpMethod:` indica el método HTTP.

`pattern:` URL del endPoint.

`script:` lógica del endPoint. Básicamente la estructura que contiene apunta a la interfaz expuesta de MetaBase explicada en la sección Herramientas utilizadas.

4.6.3. Herramienta de Administración de Tags

Para crear el lado cliente, lado servidor y configurar los parámetros que se solicitarán durante el proceso de etiquetado, se crea esta herramienta que puede accederse por medio del URL

http://host:puerto/editor/metapiTag/editor

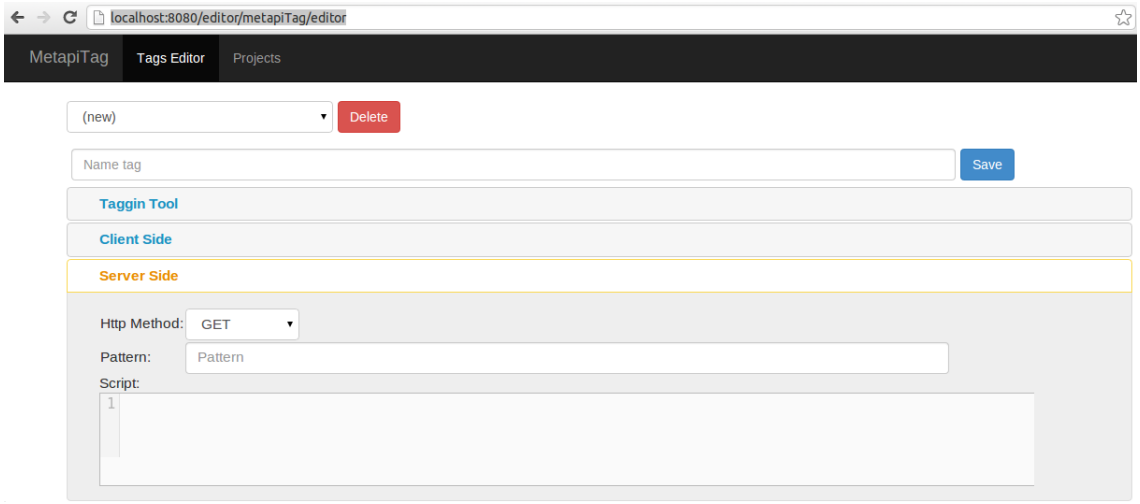


Figura 13. Herramienta de Administración de Tags.

Se observa que se encuentran tres secciones, las mismas que sirven para configurar los aspectos necesarios para el funcionamiento de un tag.

4.6.3.1. TaggingTool

En esta sección se configuran características generales del tag, como un nombre para la etiqueta del tag (`Tag Label`), se indica si el tag puede ser parte de otros tags (`Composite`) y los parámetros (`Parameters`) necesarios para el funcionamiento, tal como se indicó en la especificación de cada uno de ellos (sección ListTag), los parámetros se ingresan separados por comas. A continuación se presenta como se configuró esta sección para `ListTag`,

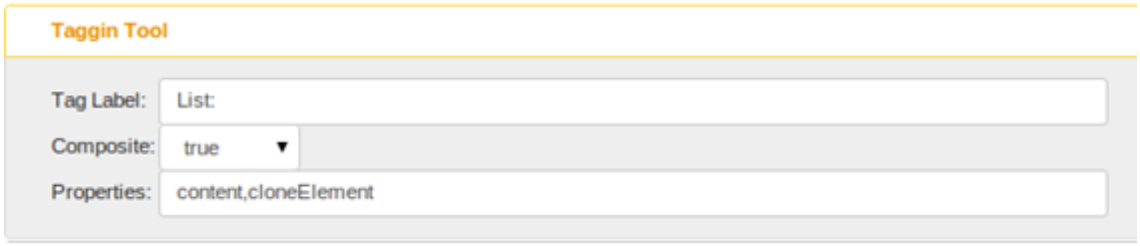


Figura 14. Configuración de TaggingTool para ListTag.

4.6.3.2. Client Side

Permite especificar la lógica del lado cliente de un tag. El código presentado a continuación corresponde a `ListTag` e indica cómo recupera la

lista de `books` (Figura 15, línea 9) y para cada elemento se agrega un elemento a la lista (Figura 15, línea 13)

Client Side

```
1  var ListTag = MetapiTag.extend({
2      init: function(selector,parameters,subTags) {
3          this.selectedObjects=[];
4          this._super(selector,parameters,subTags);
5          this.domElement=$(parameters.selector).find(parameters.cloneElement);
6      },
7      run: function(tag) {
8          var self = this;
9          self.getObjects(self, function(elements) {
10             self.removeElements();
11             $(self.domElement).remove();
12             for (i in elements) {
13                 self.addElement(elements[i],self);
14             }
15         });
```

Figura 15. Lado Cliente - ListTag

4.6.3.3. Server Side

En esta sección se configura el lado servidor de un tag si lo necesita. Para `ListTag` que requiere obtener los `books` que forman la lista, se especifica la estructura que debe tener el endPoint para soportar su funcionamiento. En esta sección para hacer referencia a los parámetros del tag se hace uso de código Mustache, la configuración del Server Side para `ListTag` se presenta a continuación y se indica los parámetros necesarios para crear el endPoint que se creará dinámicamente para obtener la lista de `books`.

Server Side

Http Method: GET

Pattern: {{&content}}

Script:

```
1 var objects=env.db.getAll('{{content}}');
2 objects;
```

Figura 16. Server Side - ListTag

4.6.4. Guardar tags para un mockup

El modelo de tags se guardará en la Base de Datos, se publica mediante un PUT al URL http:<host>:<port>/mockups/{id}/tags (Figura 17). La información a guardar será un objeto JSON que tendrá la estructura que se indicó en la sección Guardar tags para un mockup. A continuación se presenta el código que cumple el propósito indicado anteriormente.

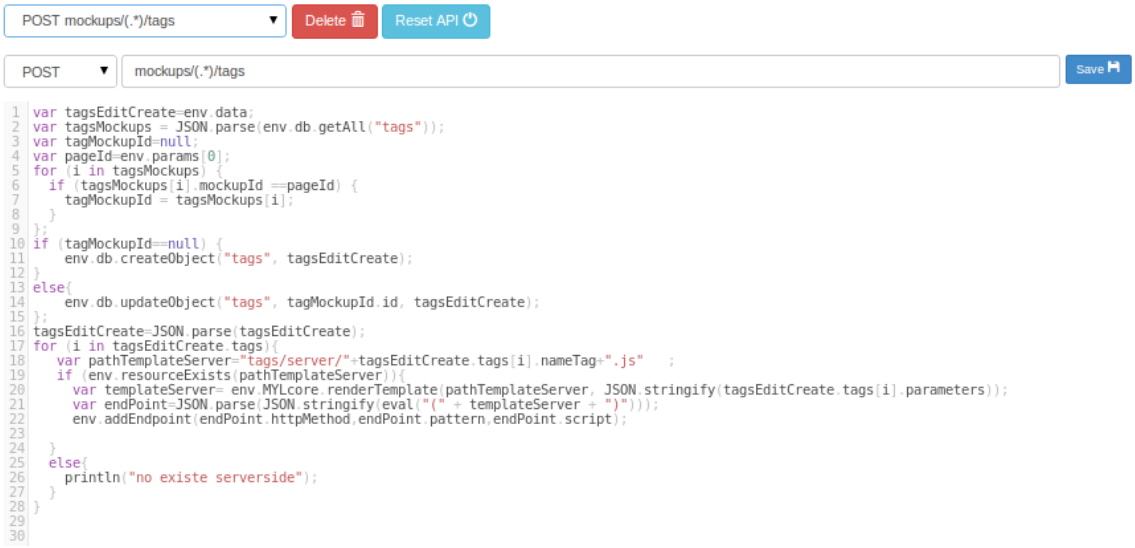


Figura 17. Código que guarda los tags de un mockup.

En resumen el código anterior, recoge los tags del mockup, que son enviados desde la herramienta, específicamente de la clase ModelStore, hace un llamado al URL indicado anteriormente y envía un objeto JSON que contiene la información de los tags a guardar (Figura 17, línea 1), el objeto es almacenado en la base de datos y para cada tag se verifica si tiene lado servidor (Figura 17, línea 19-22), si es así se crea el endPoint necesario, cuyo código se obtiene mediante código Mustache al que se le envía los datos del tag mediante un objeto JSON.

4.6.5. Herramienta de Etiquetado

Para la implementación del Módulo de Etiquetado, se crea esta herramienta, que detectará los principales elementos HTML del mockup y permitirá etiquetarlos. Para ello se forma un objeto JSON con el nombre de la página (Figura 18, línea 1) luego se utiliza taggingTool.html (Figura 19) que es una plantilla Mustache para hacer referencia a ese objeto creado. TaggingTool forma la página HTML que se renderizará, incluye el mockup como tal y todo el código necesario para que este pueda ser etiquetado, su URL es el siguiente, se indica el nombre del proyecto y mockup a ser etiquetado, y es accedido por medio de la Herramienta de Administración de proyectos que lista los mockups mediante la opción de etiquetado.

http://host:port/metapi/api/taggingTool/{projectName} /{mockupName}

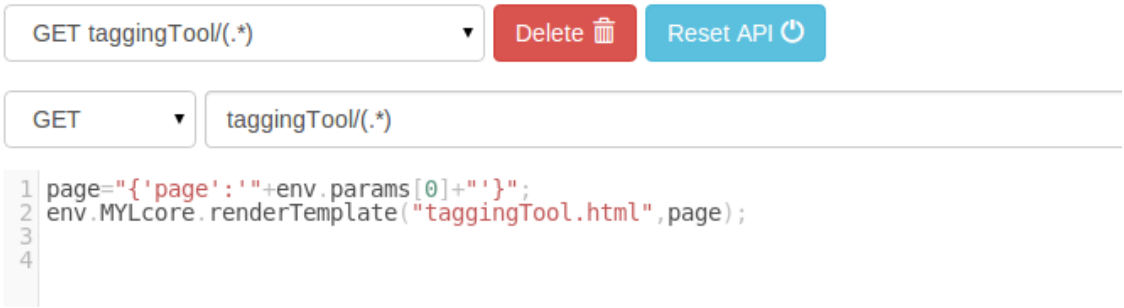


Figura 18. EndPoint TaggingTool

taggingTool.html es una plantilla Mustache guardada como recurso dentro de Metapi, incluye:

- /taggingTool/scripts/Tool.js que instancia el modelo de clases descrito en el apartado 4.5.1, el código de este script se puede visualizar en (Figura 20)
- Un iframe que hace referencia al mockup que se va a enriquecer (Figura 19, línea 4).
- La instanciación de la herramienta de Etiquetado (Figura 19, línea 12)

taggingTool.html

Save

```
<html>
<head>
    <script src="../Tool.js" type="text/javascript"></script>
    <iframe id="mockup" src="../raw/metapiTag/projects/{{&page}}"> </iframe>
<script type="text/javascript">
    $(document).ready(
            function() {
                var self = this;
                $("#mockup").load(
                    function() {
                        var rootElementP = $(this).contents();
                        Tool = new Tool(rootElementP, "metapiTag/projects/{{&page}}"
                        Tool.run();
                        $(self).contents().find("body").find("#saveButtonTaggin")
                                .click(function() {
                                Tool.saveModel();
                        });
                    });
            })
</script>
</head>
</html>
```

Figura 19. TaggingTool

```
Tool = Class.extend({

    init : function(rootElementP,mockupId) {
        this.serializer = new ModelSerializer();
        this.store = new ModelStore();
        this.model = null;
        this.editor = null;
        this.toolbox = null;
        this.rootElementP = rootElementP;
        this.mockupId = mockupId;
        this.projectId=localStorage.getItem("__project");
    },

    run : function() {
        var self = this;
        $(document).ready(function() {
            self.runNow();
        });
    },

    runNow : function() {
        var self = this;
        var modelString;
        var availableTags=self.store.getAvailableTags( function(availableTags) {
        self.store.getModel(self.mockupId, function(modelString) {
            if (!modelString) {
                self.model = new Model();
            } else {
                self.model = self.serializer.deserialize(modelString);
            }
            self.model.setAvailableTags(JSON.parse(availableTags));
            var rootElement = self.rootElementP.find("body")[0];
            self.editor = new Editor(self.model);
            self.toolbox = Toolbox.createDefault(self.model,rootElement, self.editor);

            _.each(self.model.tags, function(tag) {
                var element=$(tag.getSelector())[0];
                if (element==null)
                    element=$(rootElement).find(tag.getSelector())[0];
                tag.setModel(self.model);
                var editor = self.editor.addTag(tag, element, rootElement);
                editor.setReadMode();
            });

        });
        });
    },

    saveModel : function() {
        this.store.saveModel(this.mockupId, this.serializer.serialize(this.model, this.mockupId,this.projectId), fun
    },

    clearModel : function() {
        this.model = new Model();
        this.saveModel();
    }

});
```

Figura 20. Tool.js

El código anterior instancia la herramienta de etiquetado, mediante la función `runNow`, se obtiene los tags para ese mockup y se los asigna a la clase Model que los maneja, en caso de no poseer elementos se crea una nueva instancia.

4.6.6. Herramienta de Ejecución

Terminado el proceso de etiquetado de un mockup es posible ejecutarlo y comprobar su correcto funcionamiento, esta herramienta inyectará al archivo HTML estático, el código que instancia sus tags así como la lógica del lado cliente.

Se crea el siguiente endPoint que se accede por medio del URL

http://<host>:<port>//metapi/api/runTool/{projectName} /{mockupName}

Se crea un objeto JSON con el nombre del mockup (Figura 21, línea 1), luego se utiliza runTool.html (Figura 22) que es una plantilla Mustache y se hace referencia a ese objeto. El endPoint forma la página HTML que se renderizará, incluye el mockup como tal, la lógica del lado cliente de los tags, y el código que instancia los tags para ese mockup.

Figura 21. EndPoint runTool

runTool.html es una plantilla Mustache guardada como recurso dentro de Metapi, incluye:

- Un iframe que hace referencia al mockup que se va a enriquecer (Figura 22, línea 3).
- Dentro del iframe, se agrega los dos scripts, indicados en la sección Ejecución de tags en la plataforma.

  http://<host>:<port>/tags/client (Lógica del lado cliente de los tags disponibles) (Figura 22, línea 8)

http://<host>:<port>/mockups/{id}/tags/rendered (Código que instancia los tags que fueron etiquetados para ese mockup) (Figura 22, línea 9)

Figura 22. runTool.html

4.7. Pruebas

Con la finalidad de probar el funcionamiento de EMockupDD(metodología propuesta) y MetapiTag (herramienta de apoyo) se detalla la construcción de la aplicación ejemplo explicada en el apartado Aplicación Ejemplo, que trata de una tienda de libros el nombre del proyecto es BookStore y se podrá probar elementos básicos como listas, links, acciones CRUD.

EMockupDD inicia creando mockups necesarios para las Historias de Usuario (HU1, HU2, HU3), el modelo SUI se instancia con los widgets relevantes desde el punto de vista de las Historias de Usuario (Figura 9 a-b-c) (Paso 1 y 2. Construcción de Mockups y procesamiento). A continuación se enriquece cada mockup (Paso 3. Especificación de características.), para lograr este propósito se necesita el apoyo de MetapiTag que permite importar el proyecto BookStore con su conjunto de mockups, estilos, imágenes a la herramienta, es necesario empaquetar los archivos del proyecto en.zip o .rar y hacer uso de la Herramienta de Administración de Proyectos (Figura 23), que está disponible bajo la siguiente URL.

http://<host>:<port>/editor/metapiTag/projects

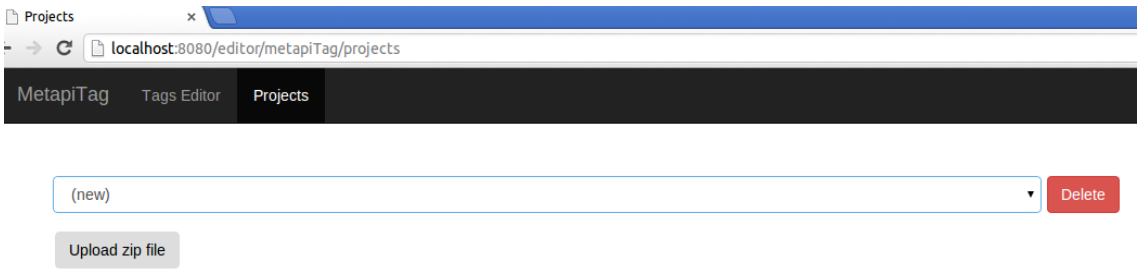


Figura 23. Herramienta Administración de Proyectos.

Posterior a la subida del proyecto se lista los mockups y para cada uno se presenta la opción de etiquetar y correr la página (Figura 24).

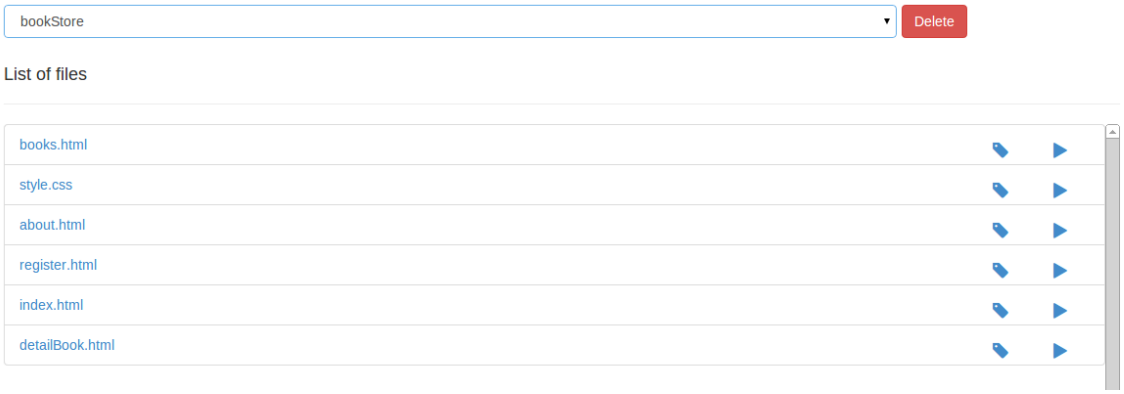


Figura 24. Lista de los mockups de un proyecto.

De la lista de mockups del proyecto BookStore, se selecciona index.html (que representa la HU1) y se elige la opción de etiquetar  que invoca a la Herramienta de Etiquetado (sección Herramienta de Etiquetado) (Figura 42, paso 3). El mockup presenta una lista de `books` disponibles con dos atributos `name` y `description`, para etiquetarlo se necesita dos tags básicos: `ListTag` que permite obtener la lista de elementos (`book`) y `PropertyListTag` para indicar los atributos a visualizar (`name y description`). Para cada atributo que se presente en la lista existirá un `PropertyListTag`, estos tags serán dependientes de `ListTag`, ya que recibirán el objeto al cual evaluarán la propiedad indicada en sus parámetros.

La Herramienta de Etiquetado, detecta los principales elementos HTML del mockup y sobre ellos es posible colocar etiquetas, las siguientes figuras indican paso a paso como crear los tags para que la lista de `books` funcione correctamente.

Figura 25. Detección del elemento lista de `books`.

Se navega por la página y se detecta el elemento que representa la lista, a continuación se da click y se presenta un menú desplegable que indica tres acciones `cancel, up, Apply Tag?` (Figura 25)

- `[cancel]:` cancela la acción de etiquetado.
- `[up]:` sirve para navegar al elemento contenedor del actual.
- `Apply Tag?:` Despliega un menú que contiene los tags disponibles. (Figura 26)

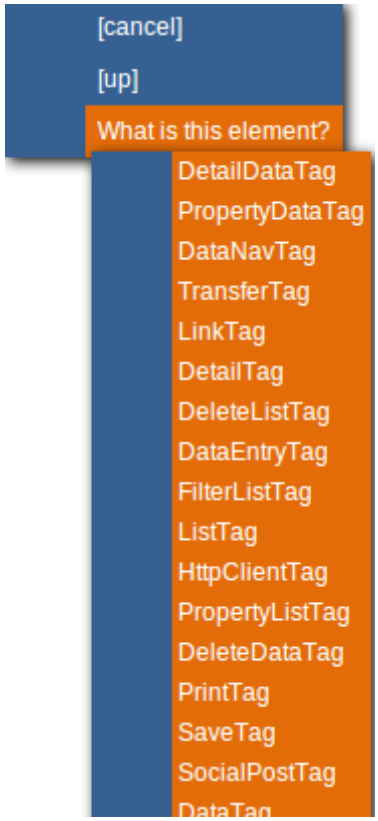


Figura 26. Menú desplegable que presenta los tags disponibles.

Se selecciona `ListTag` y se ingresa los parámetros necesarios para el funcionamiento del mismo. (Figura 27)

- Content: `book`. Indica de que elementos está formada la lista.
- CloneElement: `#book`. Selector o id del elemento de la lista que se va a repetir, para cada elemento de la lista.

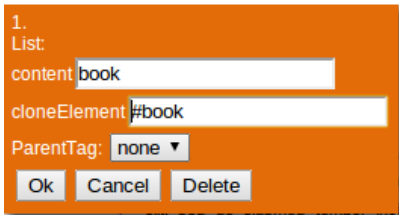


Figura 27. Configuración de lista de `books`.

A continuación para cada atributo de la lista se agrega un `PropertyListTag` que se indicará que son hijos de `ListTag` al especificar

esta relación mediante el parámetro `ParentTag` aquí se indica el id del tag padre, el mismo que es generado automáticamente por la Herramienta de Etiquetado cuando se creó `ListTag`. Para configurar que se presente el `name` de `book` se selecciona el elemento HTML correspondiente y de la lista de tags disponibles se escoge `PropertyListTag` su configuración se presenta a continuación. (Figura 28)

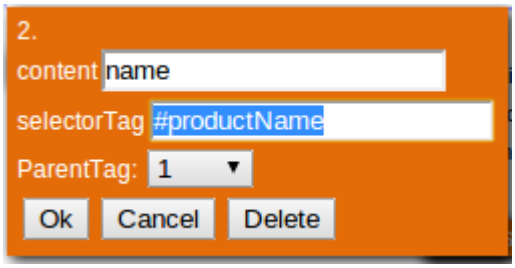


Figura 28. Configuración del atributo `name` de la lista de `books`.

El mismo procedimiento se realiza para etiquetar el atributo que presenta la descripción, una vez etiquetados todos los atributos de la lista el mockup etiquetado se ve como se muestra en la Figura 29.

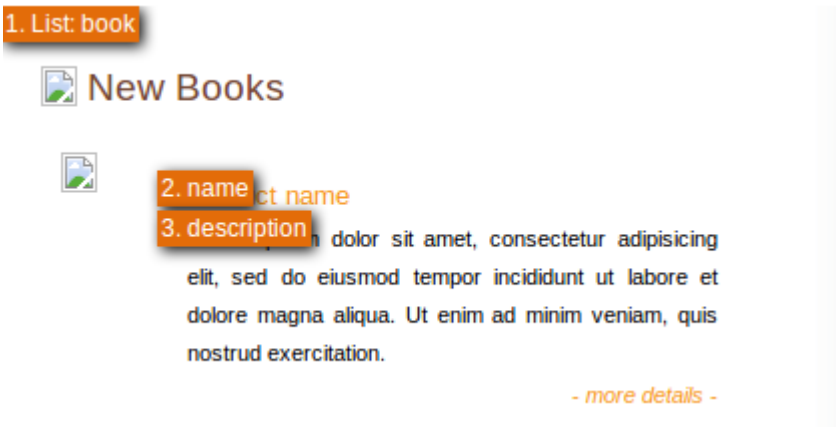


Figura 29. Etiquetado final lista de `books`.

Cuando se ha terminado la especificación de las características sobre el mockup, se guardan los tags (sección Guardar tags para un mockup), en ese momento se crea el lado servidor de los tags que lo necesiten, para el ejemplo únicamente `ListTag` necesita recuperar elementos `book` de la Base de Datos,

el endPoint creado se accede bajo el URL /`books` y la lógica que recupera la lista de libros es `env.db.getAll('book')`, todo esto se especificó al momento de crear `ListTag` (sección Herramienta de Administración de Tags)

A partir de este momento se ha inyectado dinamismo en el mockup estático, de la lista de mockups de BookStore (Figura 24), se elige la opción ejecutar (Figura 42, paso 4) para index.html y mediante la Herramienta de Ejecución (sección Herramienta de Ejecución) se puede ver corriendo la página (Figura 30).

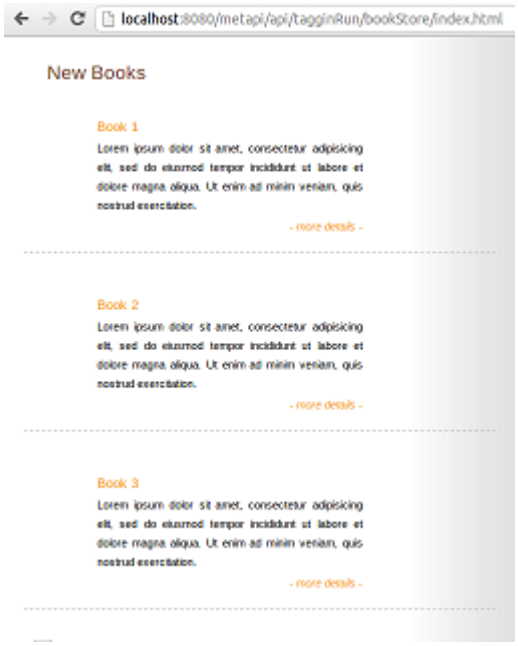


Figura 30. Aplicación corriendo. Lista de `books`

Lo descrito anteriormente indica paso a paso el proceso EMockupDD y la interacción con las Herramientas desarrolladas en MetapiTag que apoyan su desarrollo. A continuación se analiza el proceso de etiquetado de los mockups correspondientes a las HU1 y HU2.

La HU2 según lo especificado visualiza el detalle de cada `book` listado en index.html (Figura 9 - a) que representa HU1, el mockup que permite mostrar este detalle es detailBook.html (Figura 9 - b), para el proceso de etiquetado se hace uso de `LinkTag` y `TransferTag`, que re direcciona de index.html a detailBook.html.

A continuación se visualiza la configuración de `LinkTag` que se aplica sobre el elemento link de la lista de `books` de index.html (Figura 9 - a), se indica la página a la que se re direcciona, detailBook.html (Figura 9 - b) que visualizará el detalle de cada `book` y se indica que es un tag dependiente de `ListTag`, especificando el id del padre en ParentTag.

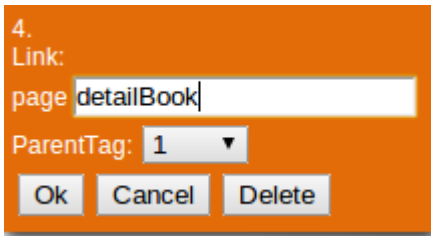


Figura 31. Configuración LinkTag.

`LinkTag` solamente re direcciona a otra página, para hacer referencia el objeto de datos a ser transferido se utiliza `TransferTag` y se indica la página y el tag al cual el objeto va a ser transportado. Es un tag dependiente de `ListTag` que permite obtener el objeto seleccionado y enviarlo a donde se necesite, para que pueda ser recuperado se hace uso de LocalStorage.

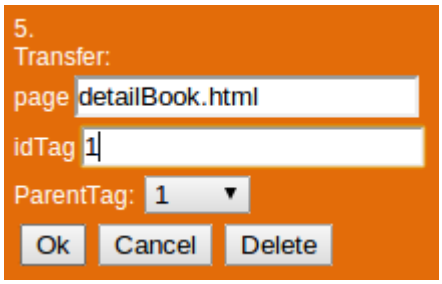


Figura 32. Configuracion TransferTag.

Los tags aplicados sobre el index.html para cubrir las HU1 y HU2 se muestran en la Figura 33.

Figura 33. Etiquetado final para la lista de `books`.

Al presionar el link de la lista de libros para ver más detalles, se re direcciona a la página configurada en `LinkTag` y se coloca en sesión el objeto que va a ser transportado a la página destino. Para HU2 se construyó detailBook.html, a continuación se presenta la imagen del resultado del proceso de etiquetar de este mockup.

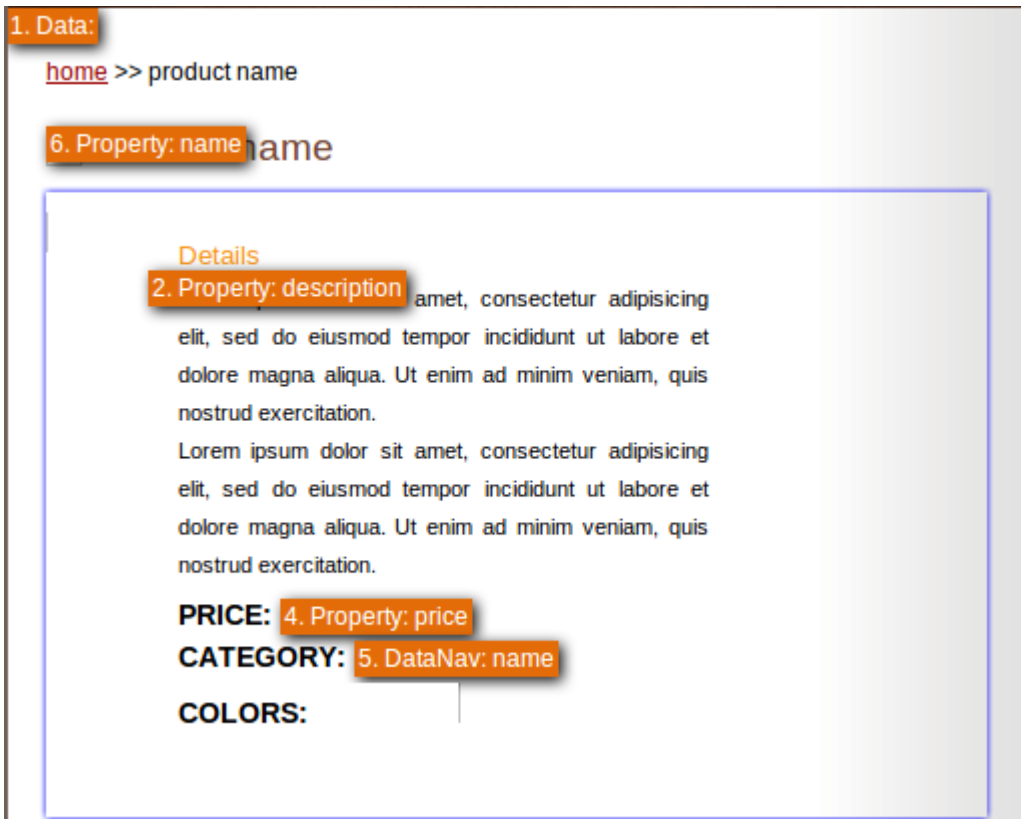


Figura 34. Etiquetado final para detalle de books.

Para etiquetar detailBook.html se hace uso de los siguientes tags `DataTag, PropertyDataTag, DataNavTag.`

- **`DataTag:`** es el tag que recibe el objeto que fue transportado desde la página index.html, la creación de este elemento es necesaria, para configurar `TransferTag` de la página principal, ya que como parámetros de entrada se necesita la página y el id del tag al que se transporta el objeto.
- **`PropertyDataTag:`** Se configuran tantos tag de este tipo como atributos necesite visualizar indicándose el nombre del atributo del objeto. Al ser un tag dependiente el padre de estos tags es `DataTag` creado anteriormente.
- **`DataNavTag:`** Este tag fue creado para cuando el contenido a visualizar pertenece a otro objeto que está relacionado con el principal, por ello se utiliza para referenciar al objeto `category` relacionado con `book.`

Para la HU3 que permite registrar un `user` dentro de la aplicación se crea register.html. A continuación se presenta la imagen del resultado del proceso de etiquetar de este mockup.

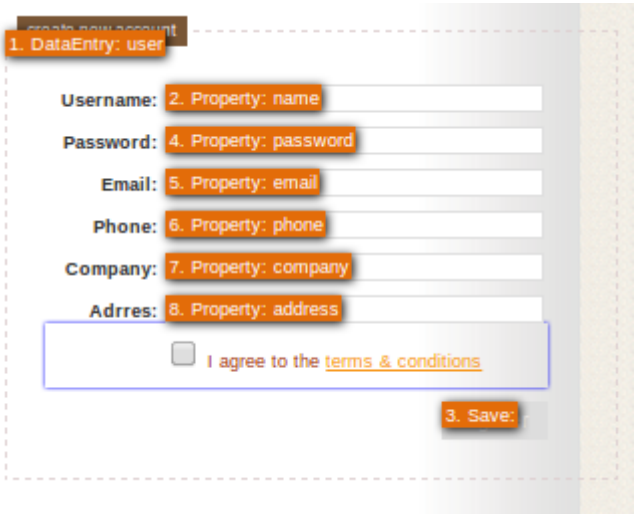


Figura 35. Etiquetado final para registro de un `user`.

- **`DataEntryTag:`** especifica el objeto que va a ser almacenado, a continuación se presenta su configuración.

Figura 36. Configuración DataEntryTag.

- **`PropertyDataTag:`** Se configuran tantos tag de este tipo, como atributos tenga el objeto a registrar, es un tag dependiente de `DataEntryTag` creado anteriormente.

- **`SaveTag:`** Este tag se encarga de ejecutar la acción de almacenamiento del objeto especificado `DataEntryTag`, que es su tag padre.

16.2 Manejo de Tags: Extensión / modificación de tags existentes

El apartado anterior indica el proceso EMockupDD desde la construcción de mockups hasta obtener una aplicación corriendo. En esta sección se cubre el proceso de afrontar un requerimiento que excede las capacidades de los tags disponibles y es una de las claves de esta investigación.

Por parte del cliente surge un nuevo requerimiento, para agregar un filtro de búsqueda sobre la lista de `books`. Para ello se crea la Historia de Usuario 4 (HU4). Se cree conveniente no crear un nuevo tag sino modificar `ListTag`. Este filtro sería aplicado a través de una consulta parametrizada que se adicionará al tag.

Para modificar un tag se utiliza la Herramienta de Administración de Tags, seleccionamos `ListTag` y se despliega las tres secciones que conforman el tag: Tagging Tool (sección TaggingTool), Client Side (sección Client Side), Server Side (sección Server Side).

16.2.1 Tagging Tool

En esta sección se agregan dos parámetros para soportar el filtro que se solicita en la HU4, esta modificación se ve reflejada en el proceso de etiquetado; cuando se soliciten los nuevos parámetros: `query` y `queryParams`.

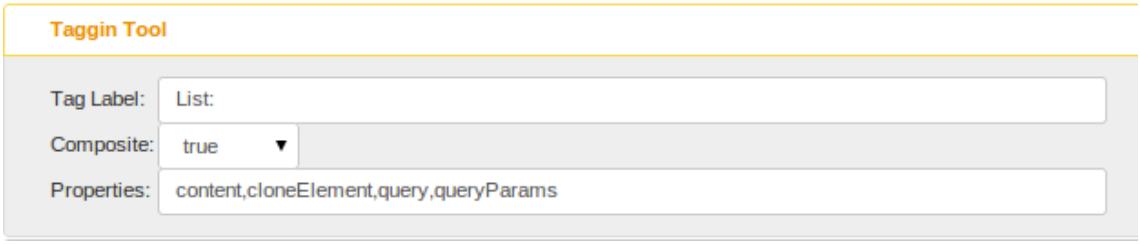


Figura 37. Nueva configuración ListTag sección Tagging Tool.

- `query`: permite ingresar el filtro para efectos de prueba se hace la búsqueda por `name`. Cuando se etiquete la lista se ingresará en este campo `o.name==:name` **o.** hace referencia al objeto de la lista `book` y `name` al atributo de `book`.
- `queryParams`: se especifica una lista de objetos JSON, cada uno de los cuales contiene la siguiente información.
    - name: indica el valor por el cual se filtra (`name`).
    - selector: id del elemento DOM que contiene ese valor (`#searchText`)
    - refresh: (`true` o `false`) indica si la lista se refresca al cambiar el valor de este elemento.
    - isParam: indica si se trata de un parámetro de búsqueda o del botón sobre el cual se aplica la acción de buscar.

Estos objetos se pueden configurar según sus necesidades, tal como se ve en el ejemplo.

```
[{'name':'name','selector':'#searchText','refresh':'true','isParam':
'true'},{'name':'','selector':'#search','refresh':'true',
'isParam':'false'}]
```

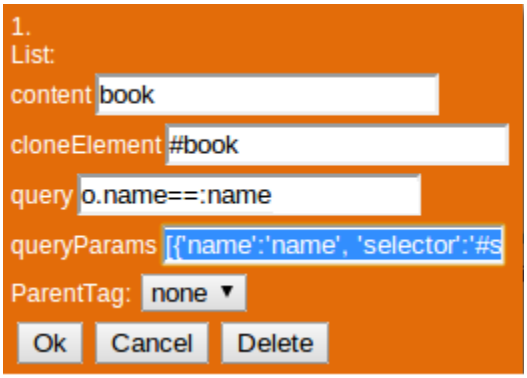
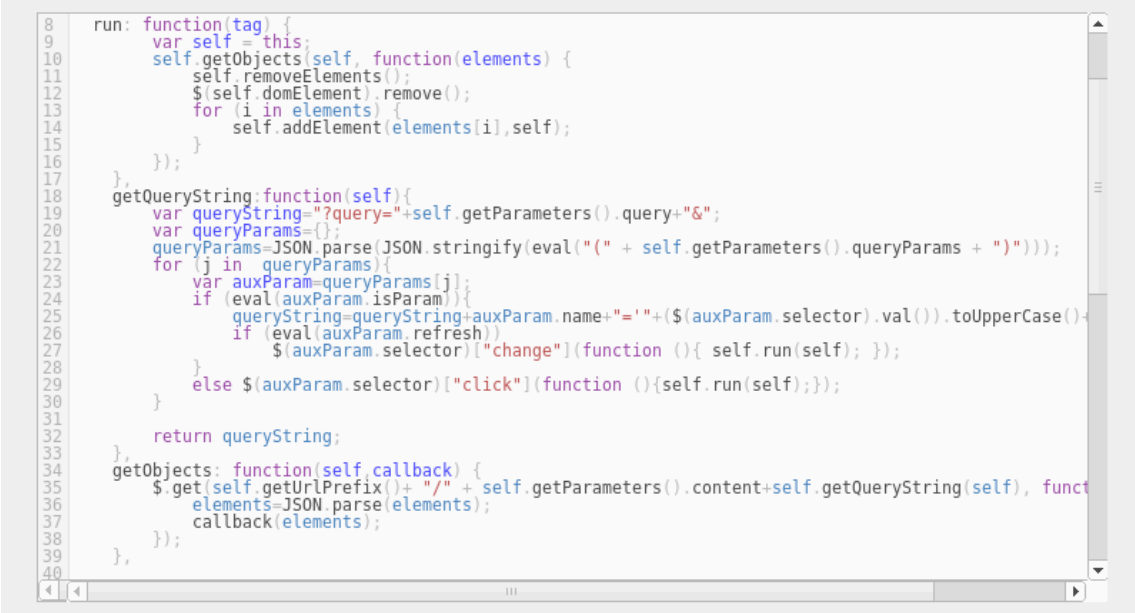


Figura 38. Nueva interfaz para etiquetar ListTag.

El paso anterior únicamente sirve para solicitar nuevos parámetros para el funcionamiento del tag en el proceso de etiquetado, la próxima vez que use este tag se solicitarán los nuevos parámetros como se ve en la Figura 38.

16.2.2 Client Side

La lógica del lado cliente del tag puede ser modificada en la sección Client Side (Client Side). El código indicado en la Figura 39 presenta los cambios realizados para soportar el nuevo requerimiento HU4. Básicamente en la línea 34 la función `getObjects()`, invoca al endPoint y lleva consigo el `query` por el cual filtrará los registros. El `query` es formado mediante la función `getQueryString`, que toma los parámetros y forma la consulta por la cual se filtrarán los objetos.



```
run: function(tag) {
     var self = this;
     self.getObjects(self, function(elements) {
         self.removeElements();
         $(self.domElement).remove();
         for (i in elements) {
             self.addElement(elements[i],self);
         }
     });
 },
 getQueryString:function(self){
     var queryString="?query="+self.getParameters().query+"&";
     var queryParams={};
     queryParams=JSON.parse(JSON.stringify(eval("(" + self.getParameters().queryParams + ")")));
     for (j in  queryParams){
         var auxParam=queryParams[j];
         if (eval(auxParam.isParam)){
             queryString=queryString+auxParam.name+"='"+($(auxParam.selector).val()).toUpperCase()+
             if (eval(auxParam.refresh))
                 $(auxParam.selector)["change"](function (){ self.run(self); });
         }
         else $(auxParam.selector)["click"](function (){self.run(self);});
     }

     return queryString;
 },
 getObjects: function(self,callback) {
     $.get(self.getUrlPrefix()+ "/" + self.getParameters().content+self.getQueryString(self), funct
         elements=JSON.parse(elements);
         callback(elements);
     });
 },
```

Figura 39. Client Side ListTag

16.2.3 Server Side

El tag requiere que los datos del lado servidor se filtren según el `query` especificado en el apartado anterior, para ello, primero se obtiene la lista de todos los `books` (Figura 40, línea 2) y luego se filtra por el `query` indicado, si el objeto cumple la condición se agrega a una lista de objetos que serán retornados (Figura 40, línea 1 - 15).

Figura 40. Server Side ListTag

La siguiente figura presenta la ejecución del mockup, con el nuevo `ListTag` aplicado.

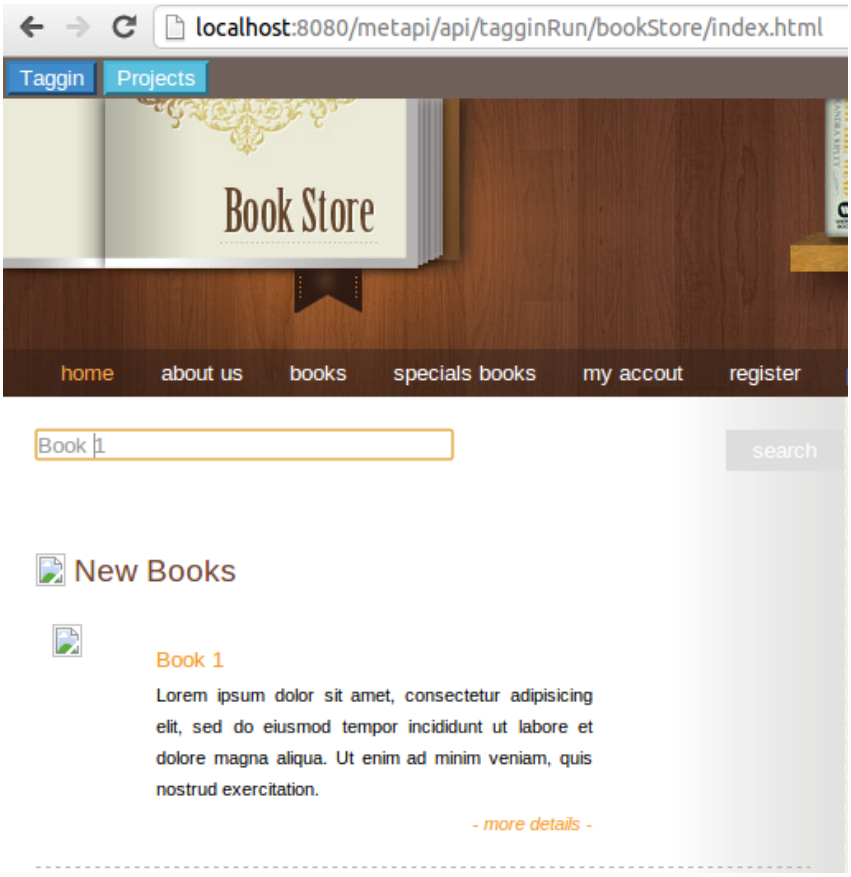


Figura 41. Página en ejecución luego de modificar ListTag

16.3 Discusión del prototipo

Los mockups que forman parte del proyecto son HTML, la posibilidad de usar los generados por herramientas digitales y ser transformados a HTML es una problemática ya abordada en MockupDD [41].

Los tags implementados son básicos y sirvieron para demostrar el proceso EMockupDD y su interacción con MetapiTag. A continuación se detalla el proceso de la metodología propuesta y su interacción con las herramientas creadas.

Figura 42. Proceso EMockupDD apoyado en MetapiTag.

EMockupDD inicia con la construcción de mockups que representan las Historias de Usuario pactadas (Figura 42, paso 1), a continuación se sube el

proyecto que contiene todos los mockups y archivos necesarios a MetapiTag haciendo uso de la *Herramienta de Administración de Proyectos* (Figura 23). Por medio de la *Herramienta de Etiquetado* (sección Herramienta de Etiquetado), el equipo de desarrollo captura los principales conceptos en los mockups construidos en un SUI meta-modelo (Figura 42, paso 2) junto con las partes interesadas interpretan la semántica en las Historias de Usuario y mockups, especifican los requerimientos, se etiquetan cada uno de ellos y se obtiene un modelo SUI completamente enriquecido (Figura 42, paso 3), hasta este punto no existe diferencias con la versión original de MockupDD.

Cuando los requerimientos exceden las capacidades expresivas de los tags se hace uso de la *Herramienta de Administración de Tags* (sección Herramienta), que permite modificar el comportamiento de los tags existentes o crear nuevos elementos.

Completado el enriquecimiento se obtiene una aplicación que puede ser evaluada de manera inmediata, a diferencia de MockupDD tradicional, los tags, no son traducidos en modelos, sino se convierten en código ejecutable que inyecta dinamismo a los prototipos estáticos (Figura 42, paso 4), esto se logra gracias a la *Herramienta de Ejecución* (sección Herramienta de Ejecución) .

17.1 CAPÍTULO 5. MEDICIÓN DE LA HERRAMIENTA

En el capítulo anterior se comentó el desarrollo de una aplicación web basada en EMockupDD, si bien se probó a nivel práctico el funcionamiento de la misma, el presente capítulo pretende llevar a cabo una evaluación de la aplicación desarrollada de dos formas distintas:

1. Recolección de las opiniones de un conjunto de desarrolladores experimentados posterior a una demostración de MetapiTag.
2. Comparación de la creación de nuevos elementos de lenguaje tanto en MetapiTag como en WebRatio[42] que es una herramienta de soporte de IFML, un estándar de modelado de aplicaciones web.

5.1 Opinión de un conjunto de desarrolladores

La evaluación de MetapiTag que es la herramienta de soporte para EMockupDD, se realizó por medio de un cuestionario enfocado a medir características que se consideran adecuadas. El cuestionario se creó para ser aplicado a un grupo de profesionales que conforman el centro de desarrollo de software de la Universidad de Cuenca (CDS). A continuación se detalla el diseño del mismo.

5.1.1 Diseño del cuestionario

El cuestionario se estructura en tres partes distintas. La primera parte tiene la finalidad de identificar el perfil del usuario de MetapiTag. Con la segunda parte, se busca identificar el grado de satisfacción del usuario y al final se realiza una pregunta abierta, para rescatar los comentarios y observaciones de los usuarios, a través de la cual se obtendrán opiniones acerca de un conjunto de mejoras para la herramienta.

Las características de cada sección del cuestionario, se detallan a continuación.

- **Evaluación del Perfil del Usuario**

La finalidad de esta sección es identificar el tipo de usuario y las metodologías de desarrollo de software con las que suelen trabajar. Esta

información permite valorar las respuestas, al considerar el nivel de experiencia del encuestado.

- **Evaluación del Herramienta**

En esta sección se evalúan características como Funcionalidad, Usabilidad, Mantenimiento, Portabilidad y Eficiencia.

- **Comentarios y aportes de los Usuarios**

Trata de recolectar las opiniones de los encuestados que sirvan para mejora el conjunto de tags disponibles.

5.1.2 Cuestionario de evaluación MetapiTag

A continuación se presenta el modelo de cuestionario, que se utilizó.

Evaluación MetapiTag						
Fecha:						
Ciudad:						
Empresa:						
Objetivo de la Herramienta: Proponer una estrategia de mejora a MockupDD agregando características adicionales al paradigma MDWE ortodoxo que implementa originalmente, que permita modificar la semántica de los conceptos de su lenguaje de tags dependiendo de un dominio específico.						
Perfil de Usuario						
Si Ud. está de acuerdo en dar su valoración, por favor conteste las siguientes interrogantes; marque con una X el espacio que considere adecuado a su apreciación, sus respuestas serán de gran valor para el desarrollo esta investigación. **¡Gracias!.**						
Edad:						
Años de experiencia desarrollando SW:						
Conocimiento de Metodologías ágiles:		SI		NO		
Conocimiento de desarrollo dirigido por modelos:		SI		NO		
Característica	**Pregunta**	**Totalmente de acuerdo**	**De acuerdo**	**Ni de acuerdo, ni en desacuerdo**	**En desacuerdo**	**Totalmente en desacuerdo**
Funcionalidad	Tiene el conjunto de funciones apropiadas para las tareas especificadas?					
	Opiniones:					
	Considera que se cumple el objetivo para el cual fue creada la herramienta?					
	Opiniones:					
Usabilidad	Es fácil entender la estructura y lógica de la herramienta?					
	Opiniones:					
	El uso de tags es intuitivo?					
	Opiniones:					
Mantenimiento	Se puede modificar el comportamiento de tags según sus necesidades?					
	Opiniones:					
	Cree necesario el conocimiento de expertos para agregar nuevos tags?					
	Opiniones:					
Portabilidad	Se pueden utilizar librerías ya desarrolladas dentro de la herramienta?					

	Opiniones:					
Eficiencia	Se obtiene una aplicación inmediatamente después de etiquetar mockups?					
	Opiniones:					
	Cree que la herramienta ayuda a desarrollar un prototipo funcional de una aplicación más rápido de lo que Ud. lo hubiera realizado utilizando codificación directa en el lenguaje y entorno en el cual se siente más productivo?					
	Opiniones:					
Según su experiencia que tags se deberían incluir en la herramienta?						

Figura 43. Cuestionario de Evaluación de MetapiTag.

5.1.3 Análisis de los resultados

Luego de aplicar la encuesta a un grupo de 6 desarrolladores pertenecientes al Centro de Desarrollo Informático de la Universidad de Cuenca se pudieron obtener los siguientes resultados.

5.1.3.1 Resultados agrupados por característica.

Los resultados presentados a continuación están agrupados por las siguientes características:

- Funcionalidad: indica si el prototipo cuenta con las funcionalidades adecuadas para cumplir los objetivos planteados.
- Usabilidad: indica la capacidad que tiene el prototipo para ser entendido, aprendido, utilizado.
- Mantenimiento: indica la capacidad que la herramienta presenta para modificar sus elementos del lenguaje y la necesidad del conocimiento de expertos para realizar esta tarea.
- Portabilidad: indica la compatibilidad de la herramienta con lógica de negocio desarrollada externa a ella.
- Eficiencia: indica si el tiempo y esfuerzo para desplegar aplicaciones es óptimo.

Figura 44. Evaluación característica Funcionalidad

Figura 45. Evaluación característica Usabilidad

Figura 46. Evaluación característica Mantenimiento

Figura 47. Evaluación característica Portabilidad

Figura 48. Evaluación característica Eficiencia

5.1.3.2 Resultados Finales

A continuación se presenta el resultado final de la valoración de MetapiTag. El resultado indica el porcentaje que cada participante dio a la herramienta. Para ello sumó el puntaje de cada pregunta y se hizo un promedio al 100 por ciento obteniendo el siguiente resultado.

Figura 49. Puntaje Total de la Herramienta por participante.

El siguiente gráfico visualiza el porcentaje de valoración de la herramienta, según la siguiente tabla.

Valoración	**Porcentaje**
Excelente	80-100
Regular	40-79
Deficiente	Menos 40

Tabla 2. Tabla valoración final herramienta

Figura 50. Valoración final herramienta. Opinión de los encuestados sobre el prototipo.

Según las opiniones que se pudieron recolectar, MetapiTag tuvo un buen nivel de aceptación, la mayoría de los participantes emitió una calificación por encima de 80/100 (Figura 49). En la característica Funcionalidad que mide criterios de cumplimiento de objetivos todos los participantes estuvieron Totalmente de acuerdo que MetapiTag cumple su propósito. Para la característica Usabilidad que indica sí MetapiTag presenta facilidad de aprendizaje y utilización, la mayoría de participantes indicó que estuvieron Totalmente de acuerdo. Para la característica Mantenimiento que mide la capacidad de agregar o modificar elementos del lenguaje, los participantes emitieron juicios igualitarios entre Totalmente de acuerdo, De acuerdo, Ni de acuerdo, ni en desacuerdo. Para la característica Portabilidad que indica la compatibilidad de MetapiTag con otras herramientas, la mayoría de los participantes dijeron que están Totalmente de acuerdo. Para la característica Eficiencia que indica si el tiempo y esfuerzo para desplegar aplicaciones es óptima la mayoría de los participantes opinó que está Totalmente de acuerdo.

Con estos resultados se llega a la conclusión de que tras una breve explicación de MetapiTag, los participantes indicaron que la herramienta cumple los objetivos para los cuales fue creada y apoya el proceso EMockupDD.

5.2 Creación de un nuevo elemento de lenguaje (Comparación WebRatio -MatapiTag)

WebRatio[42] es una herramienta que apoya la metodología MDWE con más casos de éxito a nivel industrial [43] utiliza IFML[44] como lenguaje de modelado, este se basa en pocos conceptos, que pueden integrarse y ser utilizados para ensamblar aplicaciones web complejas para la publicación o actualización de contenido.

Sin embargo, las unidades básicas previstas en IFML no son suficientes para cubrir todo el espectro de necesidades de las aplicaciones, y es posible que se desee utilizar unidades implementadas por los desarrolladores. Para este propósito, WebRatio incluye el concepto de unidades personalizadas que son definidas por el programador, y no se encuentran incluidas en el lenguaje IFML estándar.

MetapiTag ofrece un comportamiento similar, basándose en un lenguaje de tags los cuales pueden agruparse y lograr comportamientos complejos. Sin embargo como ya se explicó al comienzo, las necesidades particulares de las aplicaciones no siempre son cubiertas y es por ello que se vislumbra la necesidad de crear nuevas funcionalidades.

A continuación se realiza una comparación del esfuerzo que es necesario realizar para crear nuevos elementos de lenguaje personalizados tanto en WebRatio como MetapiTag.

Se propone crear dos componentes: un cliente HTTP que por medio de un URI se identifique atributos como Host, Method y Port. Un widget que permita hacer un de comentario en Facebook desde un link.

5.2.1 Creación cliente HTTP

5.2.1.1 Cliente HTTP - WebRatio

El desarrollo de una unidad personalizada permite agregar nuevos componentes de la arquitectura WebRatio. La definición y ejecución de una unidad necesita abordar aspectos tanto en tiempo de diseño y las cuestiones de tiempo de ejecución.

El primer paso para desarrollar una unidad personalizada es crear un nuevo proyecto de Componentes. File> New> Component Projects sirve para crear un nuevo proyecto de componentes. Ingresamos `MyCustomUnits` para el nombre del contenedor de unidades personalizadas. File> New> Component sirve para crear una nueva unidad personalizada. El nombre para esta unidad es `HttpClientUnit`. Luego de crear la unidad, se presenta una estructura en árbol de carpetas. Cada carpeta contiene archivos asociados a un aspecto diferente de la especificación de la unidad.

1. **Unit.xml**: Contiene la definición de la unidad y es utilizada para permitir la edición de la misma en el modelo de web, y mostrar sus propiedades, que pueden ser ingresadas en la pestaña View

.

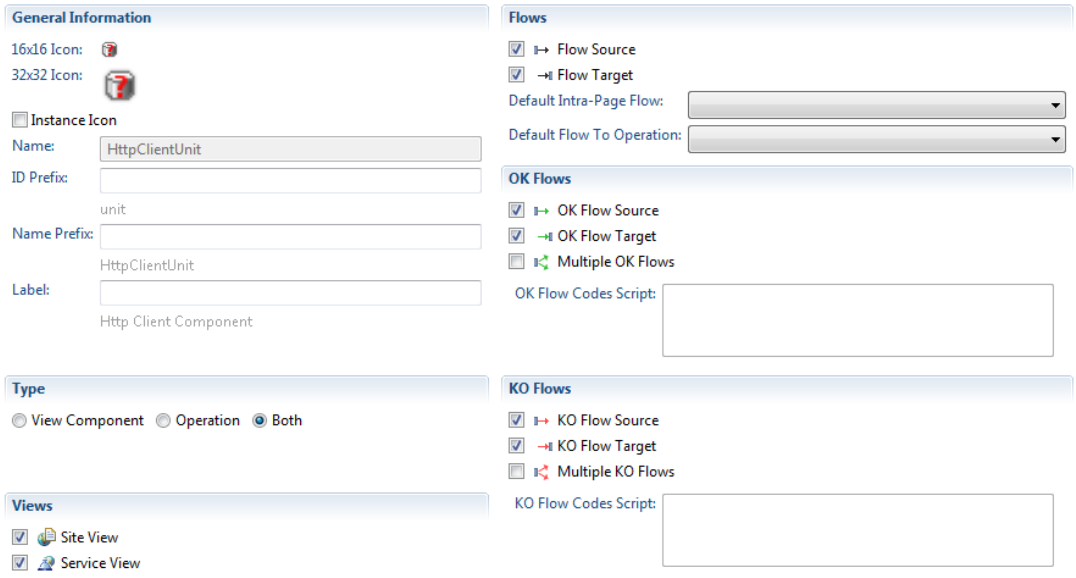


Figura 51. Unit.xml para crear HttpClientUnit

Los parámetros de entrada y salida se indican en la pestaña de Sub-Elements, a continuación se presenta la configuración de los mismos. Estos parámetros se utilizan para obtener información del URI que ingresa.

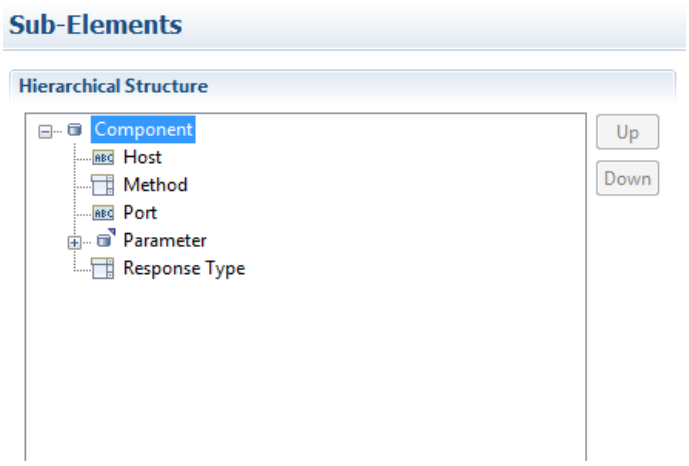


Figura 52. Interfaz para indicar parámetros de entrada salida de un nuevo elemento WebRatio.

2. **Imágenes**: Contiene las imágenes utilizadas para representar la unidad en el modelo Web.

La lógica de la unidad está compuesta por tres archivos; dos de ellos (Output.template, Input.template) especifican parámetros de entrada y salida y un archivo (llamado Logic.template) produce en tiempo de ejecución un descriptor XML para el servicio de la unidad.

3. **Input.Template:** A continuación se observa el código necesario para configurar un parámetro de entrada a la Unidad, el parámetro es URI y se indica un nombre, un tipo y una etiqueta. El código está escrito en lenguaje Groovy.

```
#?delimiters <%,%>,<%=,%>
 <%
 setXMLOutput()
 def unitId = unit.valueOf("@id")
 def params = unit.selectNodes("Parameter")
 %>
 <InputParameters>
    <InputParameter name="<%= unitId%>.uri" type="" label="URI"/>
    <% for (param in params) { %>
        <InputParameter name="<%= unitId%>.<%= param["name"]%>" type="" label="<%= param["name"]%>"/>
    <% } %>
 </InputParameters>
```

Figura 53. Código para configurar parámetros de entrada del nuevo elemento WebRatio.

4. **Output .Template:** archivo de configuración para indicar los parámetros de salida (Host, Method, Port), igual que el anterior es un fragmento de código Groovy.

```
#?delimiters <%,%>,<%=,%>
<%
setXMLOutput()
def unitId = unit["id"]
def params = unit.selectNodes("Parameter")
%>
<Descriptor service="com.webratio.units.community.net.HttpClientUnitService">
   <Host><%= unit["host"]%></Host>
   <Port><%= unit["port"]%></Port>
   <Method><%= unit["method"]%></Method>
   <ResponseType><%= unit["responseType"]%></ResponseType>
   <Parameters>
      <% for (param in params) { %>
        <Parameter><%= param["name"]%></Parameter>
      <% } %>
   </Parameters>
</Descriptor>
```

Figura 54. Código para configurar parámetros de salida del nuevo elemento WebRatio.

5. **Advertencias:** archivo llamado WebModel.template utilizado para mostrar los errores y advertencias en el uso de la unidad

```
#?delimiters <%,%>,<%=,%>
<%
    if (unit["host"] == "") {
         addWarning(unit, true, "Host is unspecified.")
     }
     if (unit["port"] == "") {
         addWarning(unit, true, "Port is unspecified.")
     }
     if (unit["method"] == "") {
        addWarning(unit, true, "Method is unspecified.")
     }
     if (unit["responseType"] == "") {
        addWarning(unit, true, "Response type  is unspecified.")
     }
%>
```

Figura 55. Código para configurar errores y advertencias del elemento WebRatio.

6. **Java – Runtime Service**

La clase Java define el comportamiento en tiempo de ejecución de la unidad tiene la siguiente estructura.

```
public HttpClientUnitService(String id, RTXManager mgr, Element descr)throws RTXException {
    super(id, mgr, descr);
    host = DescriptorHelper.getChildValue(descr, "Host", true, this);
    port = DescriptorHelper.getChildValue(descr, "Port", true, this);
    method = DescriptorHelper.getChildValue(descr, "Method", true, this);
    responseType = DescriptorHelper.getChildValue(descr, "ResponseType",
            true, this);
    Element parametersElement = DescriptorHelper.getChild(descr,
            "Parameters", true, this);
    parameters = DescriptorHelper.getChildValues(parametersElement,
            "Parameter", false, this);
}
```

Figura 56. Clase Java del comportamiento del nuevo elemento WebRatio.

El primer argumento, es el identificador único de la unidad. El tercer argumento del constructor es un elemento DOM4J, asociado con el elemento raíz del descriptor de lógica de la unidad. El descriptor de la lógica se accede a través de las API dom4j estándar o con algunos métodos de utilidad que proporciona la clase com.webratio.rtx.core.DescriptorHelper.

5.2.1.2 ClienteHttpTag - MetapiTag

El desarrollo de un tag personalizado al igual que en WebRatio cubre toda la arquitectura de un tag dentro de MetapiTag. Ya que la definición y la ejecución de cada tag aborda aspectos de diseño (Etiquetado TaggingTool TaggingTool) y cuestiones de tiempo de ejecución (lado cliente, sección Client Side Client Side – lado servidor, sección Server Side Server Side).

El primer paso para desarrollar un tag personalizado es acceder a la interfaz destinada para este fin (Herramienta de Administración de Tags), en el capítulo anterior se explicó de manera detallada los pasos a seguir para modificar de un tag. A continuación se presentan el proceso a seguir para crear HttpClientTag.

1. **Configuración inicial:** Se ingresa el nombre del tag, y se editan los campos necesarios su funcionamiento. Los parámetros de entrada al tag se indican en el campo properties, para el ejemplo a crear se tiene un parámetro de entrada llamado URI, en caso de ser necesario más parámetros se los separa con comas.

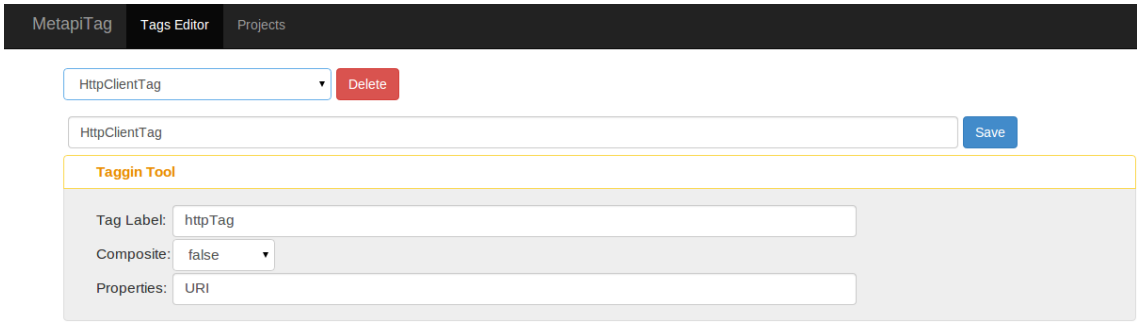


Figura 57. Configuración inicial de HttpClientTag.

2. **Lado Cliente:** Se edita la lógica del cliente del tag. La lógica (cual se muestra a en la Figura 58) invoca a un URL llamado /httpClient que representa la funcionalidad del tag.

Figura 58. Configuración del lado cliente de HttpClientTag.

3. **Lado Servidor:** En el apartado anterior se indicó que el lado cliente invoca el endPoint que se crea en base a la especificación de esta sección, a continuación se visualiza el fragmento de código necesario para invocar a una clase en tiempo de ejecución. La clase a la que se invoca se ha empaquetado dentro de un jar, y no contiene librerías especiales, sólo las librerías propias para el funcionamiento de HttpClient.

Metapi ha sido modificada para permitir que una clase pueda ser instanciada y sea posible importar paquetes ya desarrollados.

```
env.helper(<<library.jar>>,<<clase>>)
```

Esta sentencia devuelve un objeto que representa una instancia del helper solicitado, lo cual permite invocar sus métodos. La Figura 59 indica la lógica del lado servidor en donde se instancia el helper httpClient (Figura 60), se invoca el proceso execute para luego obtener el nombre del host perteneciente al URI especificado.

Para que el helper pueda ser utilizado dentro de MetapiTag, se crea una carpeta que empaquete el archivo jar y las dependencias necesarias para su correcto funcionamiento y se almacena como recurso dentro del API.

Figura 59. Configuración del lado servidor de HttpClientTag.

```
package client;
import java.io.IOException;
 public class Cliente {
    private String URI;
    private String host;
    private String method;
    private int port;

    public void execute(String uri){
         try {
            HttpClient client = createHttpClient(host, port);
            GetMethod httpget = new GetMethod(uri);
            client.executeMethod(httpget);
            this.port=httpget.getHostConfiguration().getPort();
            this.host=httpget.getHostConfiguration().getHost();
            this.method=new GetMethod(uri).getName();
        }
        catch(Exception e){ }
    }

    private HttpClient createHttpClient(String ip, int port) {
    private Object executeMethod(String methodType, HttpClient client, String uri,
    public void dispose() {
    public Cliente(){
 }
```

Figura 60. Helper HttpClient

5.2.2 Creación PostSocialUnit

En esta sección se analizará la introducción de un elemento en ambas propuestas. El nuevo elemento que se incorporará, permite enviar comentarios a la red social Facebook.

5.2.2.1 SocialUnitPost - WebRatio

A continuación se presenta la configuración de cada una de las secciones necesarias para que la unidad funcione correctamente.

1. **Unit.xml**: se indica que el tipo de la Unidad es de Operación ya que no devuelve ningún contenido.
2. **Imágenes**: Se hace referencia al ícono de Facebook.
3. **Input.Template:** se configura un parámetro de entrada `FBApplicationID`, con el que se registra a la red social y permite realizar el comentario.
4. **Output .Template:** No tiene parámetros de salida.
5. **Advertencias:** se configura errores y advertencias en el uso de la unidad, para el caso de que no se ingrese el parámetro `FBApplicationID`.
6. **Java – Runtime Service**

Una parte muy importante de la Unidad es la clase donde toda la funcionalidad de la unidad está implementada. Se configura el método `connectFB` () que obtiene los datos de identificación de la aplicación de Facebook. `OperationContext` se obtiene cuando el método execute ha sido llamado y es una implementación propia de WebRatio para acceder al parámetro de entrada `FBApplicationID`. Se crea una nueva instancia de `FacebookJSONRestClient` con la identificación de la aplicación, este API permite realizar el comentario en la red social.

```
private FacebookJSONRestClient connectFB (Map operationContext){
    String
    FB_APP_API_KEY=BeanHelper.asString(operationContext.get("FBApplicationID"));
    return new FacebookJSONRestClient(FB_APP_API_KEY);
}
```

El método `sendFB` crea la instancia de `FacebookJSONRestClient` y utiliza `connectFB` () y `stream_publish` (), para realizar el funcionamiento requerido.

```
private void sendFB (String message, Map operationContext, Long userID) throws
FacebookException  {

    FacebookJSONRestClient facebookClient=connectFB(operationContext);
    facebookClient.stream_publish(null,null,null,null,userID);
}
```

5.2.2.2 SocialPostTag – MetapiTag


```
var SocialPostTag = MetapiTag.extend({

    run: function(tag) {
        var self = this;
      $(self.getParameters().selector)["click"](function(e){      publish();} )
            $('document').ready(function(){

                $(this).contents().find("head").
                append('<script>  window.fbAsyncInit = function() {    '+
                       'FB.init({  appId: "{{appId}}", status: true, cookie: true,xfbml: true});};'+
                       'var publish = function () {    '+
                            'FB.publish({    },function(published){      '+
                       'if (published)       alert("publicado!");     '+
                       'else alert("No publicado.");    }); |}(document));</script>');
            });
```

Figura 61. Configuración SocialPostTag

1. **Configuración Inicial:** Se ingresa el nombre del tag, y se editan los campos necesarios para el funcionamiento de la herramienta de etiquetado, los parámetros de entrada al tag se indican en el campo properties, en este caso se tiene un parámetro de entrada llamado `appId`, que servirá para la configuración de la conexión al API de Facebook.
2. **Lado Cliente:** Se edita el lado cliente con la lógica del comportamiento del tag, y se invoca al API de Facebook (Figura 61).
3. **Lado Servidor:** No contiene lado servidor, ya que el API utilizada es manejada totalmente en el lado cliente.

5.2.3 Resumen. Creación unidad personalizada WebRatio

La adición de una unidad a medida en WebRatio requiere:

1. Añadir una definición de la unidad para la biblioteca de unidades (obligatorio). El archivo Unit.xml puede ser modificado por medio del editor de herramienta, para crear los comandos apropiados y colocar la Unit personalizada en el Modelo Web.
2. Indicar las imágenes utilizadas para representar la unidad en el Modelo Web.
3. Adición de un archivo Input.template, (plantilla Groovy) para indicar parámetros de entrada a la unidad.
4. Adición de un archivo Output.template, (plantilla Groovy) para indicar parámetros de salida a la unidad.
5. Adición de WebModel.template, plantilla con un fragmento de código Groovy, utilizada para mostrar errores y advertencias.
6. La implementación de una clase que posterior a su creación, se la importe en tiempo de ejecución y hace referencia a lo que en realidad realiza el servicio de negocio para el que la unidad está diseñada.

WebRatio permiten a los desarrolladores usar sus componentes en el modelo Web de cada Proyecto Web y estos componentes cooperarán con las unidades estándar.

5.2.4 Resumen. Creación tag personalizado MetapiTag

La adición de un tag personalizado se realiza mediante la Herramienta de Administración de Tags, en donde, se presenta tres secciones a editar.

1. TagginTool: se ingresa información del nombre del tag, y los parámetros que se solicitarán en el momento de etiquetar un mockup. (Figura 61, sección Tagging Tool)
2. Client Side: se escribe el código de comportamiento del tag en el lado cliente.
3. ServerSide: se registra el comportamiento del tag en el lado servidor, escribiendo los endpoints necesarios en la API RESTful asociada al tag.

Si se necesita incorporar una clase en tiempo de ejecución que ofrezca funcionalidades necesarias, puede ser agregada al proyecto. Es necesario crear

una carpeta con el nombre del .jar e incluir todas las dependencias necesarias e importar esta carpeta a Metapi como recurso.

Podemos hacer uso de las clases de este paquete, mediante.

```
Class clase=env.helper(library,_class);
clase.method(inputParams);
```

5.2.5 Análisis de los resultados

Se han creado dos nuevos elementos de lenguaje, tanto en WebRatio como en MetapiTag, a continuación se realiza una comparación de este proceso en las dos herramientas.

	WebRatio	**MetapiTag**
Pasos requeridos	6	3
Necesidad incluir librerías especiales	Si	No
Conocimiento de lenguaje especial	Si	No
Conocimientos API a utilizar	Si	Si
Configuración en una sola interfaz	No	Si

Tabla 3**. Comparación WebRatio-MetapiTag. Creación de un nuevo elemento de lenguaje.**

La Tabla 3 indica que los pasos requeridos para crear un nuevo elemento de lenguaje en MetapiTag es menor que en WebRatio. Las clases implementadas contienen las librerías necesarias para su funcionamiento, más en WebRatio se requiere agregar librerías especiales para hacer referencia a los parámetros de entrada, salida. Si bien ambos enfoques obligan a codificar, en MetapiTag se trabaja con java puro y luego con js interpretado y no es necesario conocer Groovy ni todos los templates y xmls de WebRatio. Para ambos enfoques se necesitó codificar la lógica de cada elemento. MetapiTag permite la configuración de cada tag en una sola interfaz un Unit en WebRatio se configura en diferentes archivos.

18.1__

19.1 CAPÍTULO 6. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

MockupDD propone un enfoque ágil para MDWE partiendo de mockups para facilitar la generación de prototipos y modelos de software, al mismo tiempo, presenta las mismas dificultades que los procesos MDWE convencionales durante el desarrollo. Tratar de implementar requerimientos específicos que excedan las capacidades del lenguaje implica: agregar nuevos elementos al metamodelo y modificar generadores de código.

Producto de esta investigación surge EMockupDD cuyo objetivo es solucionar los inconvenientes comentados para MockupDD, a través del proceso propuesto y el uso de MetapiTag que apoya su proceso para permitir agregar o modificar tags de manera simple.

Agregar características a MDWE puro que utiliza MockupDD refinó los elementos esenciales de su lenguaje (tags), de conceptos inmutables a prototipos de implementaciones que pueden ser modificados según se requiera utilizando codificación manual. En este contexto, cada tag es tratado como una fracción de código ejecutable, tanto en sus facetas cliente como servidor.

Tras una descripción de cuáles han sido los aspectos clave que se han estudiado dentro de la investigación, en el siguiente apartado se expone el grado de cumplimiento de los objetivos planteados.

6.1 Verificación y evaluación de los objetivos

En la introducción de la tesis se planteó el siguiente objetivo principal:

Proponer una estrategia de mejora a MockupDD al agregar características al paradigma MDWE ortodoxo que implementa originalmente, que permita modificar la semántica de los conceptos de su lenguaje dependiendo de un dominio específico.

Este objetivo se ha dividido en una serie de objetivos parciales, cuya verificación, evaluación y grado de cumplimiento se comentan a continuación.

- Realizar un estudio del estado del arte en relación a metodologías ágiles y MDWE para determinar el estado actual y ahondar en las limitaciones que

presentan, el cual fue abordado en el capítulo 2 se llevó a cabo el estudio sobre estas metodologías.

- Plantear una arquitectura que permita superar las dificultades inherentes al paradigma MDWE puro de MockupDD, al tratar los elementos de lenguaje como prototipos de implementación y no como conceptos estáticos e inmutables. Siendo este el objetivo más importante, tras el estudio de las áreas mencionadas en el primer objetivo, en el capítulo 3 se describe una arquitectura que permite superar las dificultades encontradas, esta arquitectura permite extender MockupDD al permitir modificar y agregar tags sin mayores complicaciones.
- Implementar un prototipo enmarcado dentro de la arquitectura propuesta cuya implementación fue detallada en el capítulo 4
- Evaluar la estrategia propuesta al validar el funcionamiento de la herramienta (MetapiTag), tomando en cuenta variables como tiempos de desarrollo, aprendizaje y ajuste a requerimientos específicos.

El capítulo 6 describe la creación de un cuestionario que se aplicó a un conjunto de desarrolladores para recoger sus opiniones y sugerencias, luego de una presentación del prototipo. En esta misma sección se planteó una validación adicional en la cual se crearon nuevos elementos de lenguaje tanto en MetapiTag como en WebRatio. El objetivo fue realizar una comparación y comprobar que en MetapiTag se superaron las dificultades de las metodologías que siguen el flujo MDWE clásico para extender su lenguaje y demostrar las ventajas de la misma.

6.2 Aportaciones de la Tesis

El resultado de la investigación y desarrollo llevados a cabo en esta tesis tiene como resultado una metodología (EMockupDD) que mejora MockupDD en los siguientes aspectos:

- A partir de mockups enriquecidos, MockupDD genera modelos derivables a versiones ejecutables de la aplicación web modelada (por ejemplo, modelos WebML o UWE). El trabajo propuesto ya no generan modelos, en su lugar, se obtiene la aplicación ejecutable a partir de los tags, a pesar de ello conserva la característica de

abstracción ya que los tags son presentados en un lenguaje de alto nivel.

- MockupDD está restringida a las especificaciones del lenguaje de los modelos que utilice. Al igual que en metodologías MDWE convencionales como WebML o UWE, tratar requerimientos que excedan las capacidades expresivas del lenguaje utilizado presenta dificultades ya comentadas. La utilidad de esta investigación se encuadra en permitir de manera sencilla y ágil agregar nuevos conceptos de lenguaje utilizando codificación manual, así como modificar los existentes sin requerir conocimientos avanzados de modelado.

MetapiTag permitió validar EMockupDD y según las opiniones de un conjunto de desarrolladores con 3 años de experiencia promedio, se comprueba que la herramienta cumple los propósitos para los que fue creada.

Al comparar MetapiTag con WebRatio, se concluye que la principal ventaja de agregar nuevos elementos de lenguaje no requiere más conocimientos que JavaScript y otros lenguajes básicos relacionados con el desarrollo web (como HTML, CSS, etc.). Por otro lado, WebRatio requiere para extender su lenguaje conocimiento de varios lenguajes de markup y librerías específicas.

6.3 Futuras líneas de trabajo

Debido a lo extenso del área que se ha tratado surgen las siguientes líneas de investigación:

- La herramienta contemplada sólo incluye algunos tags básicos para demostrar cómo puede modelarse e implementarse aplicaciones web convencionales. Sin embargo, existen gran cantidad de tags por definirse los cuales pueden implementar diferentes funcionalidades asociadas a requerimientos de aplicaciones web actuales. Tomando en cuenta las opiniones de los desarrolladores que validaron la herramienta se sugiere desarrollar tags para cubrir aspectos no tratados aquí como por ejemplo flujo de procesos y acceso a redes sociales.
- Proponer más clases Helper que ayuden con funcionalidad básica.

- Realizar experimentos para probar la metodología en otros desarrollos.
- Deployments automáticos: actualmente se modela una aplicación en un solo servidor. Se debería soportar múltiples aplicaciones corriendo en producción y permitir trasladar una nueva funcionalidad a todas ellas.

20.1 ACRÓNIMOS

MDWE	Model Driven Web Engineering
MockupDD	Mockup-Driven Development
MDD	Model Driven Development
MDA	Model Driven Architecture
OMG	Object Management Group
CIM	Computation Independent Model
PIM	Platform Independent Model
PSM	Platform Specific Model
AMDD	Agile Model Driven Development
OOHDM	Object-Oriented Hypermedia Design Method
UWE	UML-Based Web Engineering
WebML	Web Modeling Language
IFML	Interaction Flow Modeling Language
SUI	Structural User Interface
MODFM	Mockup-DrivenFast-prototyping Methodology
JS	JavaScript
JSON	JavaScript Object Notation
DOM	Document Object Model

21.1

22.1 BIBLIOGRAFÍA

[1] J. Sutherland and K. Schwaber, “The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process.” .

[2] G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, “Modeling and Implementing Web Applications using OOHDM,” in *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, Eds. London: Springer London, 2008, pp. 109–155.

[3] J. M. Rivero, G. Rossi, J. Grigera, E. R. Luna, and A. Navarro, “From Interface Mockups to Web Application Models.”

[4] “OMG Model Driven Architecture.” [Online]. Available: www.omg.org/mda/. [Accessed: 15-Jan-2014].

[5] “Manifesto for agile software development.”

[6] C. Pons, R. Giandini, and G. Pérez, *Desarrollo de Software dirigido por modelos.* 2010, p. 280.

[7] G. Rossi, O. Pasto, D. Shwabe, and L. Olsina, *Web Engineering. Modeling and Implementing Web Applications*. .

[8] E. D. L. L, M. G. G, M. L. S, and E. L. I. R, “Proceso de Desarrollo de Software Mediante Herramientas MDA,” pp. 6–10, 2006.

[9] “OMG.” [Online]. Available: http://www.omg.org. [Accessed: 20-Jan-2014].

[10] N. Jose, “Agile Modeling,” 2010.

[11] S. W. Ambler, “Agile Model Driven Development (AMDD),” no. February, pp. 13–21, 2007.

[12] W. R. Gerti Kappel, Birgit Pröll, Siegried Reich, *Web Engineering: The Discipline of Systematic Development of Web Applications*. 2006.

[13] S. Ceri, P. Fraternali, and A. Bongio, “Web Modeling Language (WebML): a modeling language for designing Web sites,” *Comput. Networks*, vol. 33, no. 1–6, pp. 137–157, Jun. 2000.

[14] “UWE – UML-based Web Engineering.” [Online]. Available: http://uwe.pst.ifi.lmu.de/index.html. [Accessed: 12-Mar-2014].

[15] I. Jacobson, G. Booch, and R. James, *The Unified Software Development Process*, Addison We. 1999.

[16] B. Grady, R. James, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison Wesley (1999), 1999.

[17] “http://argouml.tigris.org/.” .

[18] “MagicUWE.” [Online]. Available: http://uwe.pst.ifi.lmu.de/toolMagicUWE.html.

[19] A. Wesley, *Use Cases: Requirements in Context*. 2004, p. 243.

[20] “webML.” [Online]. Available: http://www.webml.org. [Accessed: 07-Apr-2014].

[21] “Agile Alliance.” [Online]. Available: http://www.agilealliance.org/.

[22] “State of agile survey.” .

[23] M. Fowler, “The new methodology.” [Online]. Available: http://www.martinfowler.com/articles/newMethodology.html.

[24] K. M. Calo, E. Estevez, and P. Fillottrani, “Evaluación de Metodologías Ágiles para Desarrollo de Software,” pp. 455–459, 2010.

[25] “Proyectos Agiles.” [Online]. Available: http://www.proyectosagiles.org.

[26] B. C. Labrin, “Métodos Ágiles como Alternativa al Proceso de Desarrollo Web,” pp. 1086–1098.

[27] 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.

[28] F. Ricca, “Assessing the Effect of Screen Mockups on the Comprehension of Functional Requirements,” vol. V, no. 212, 2001.

[29] A. Ravid and D. M. Berry, “A Method for Extracting and Stating Software Requirements that a,” 2000.

[30] G. Reggio, F. Ricca, M. Leotta, and U. Genova, “Improving the Quality and the Comprehension of Requirements : Disciplined Use Cases and Mockups.”

[31] and E. A. F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, “On the effort of augmenting use cases with screen mockups: results from a preliminary empirical study,” 2010.

[32] “Pencil Project.” [Online]. Available: http://pencil.evolus.vn/.

[33] J. M. Rivero, G. Rossi, J. Grigera, J. Burella, E. R. Luna, and S. Gordillo, “From Mockups to User Interface Models: An Extensible Model Driven Approach,” *Design*, vol. 6385, pp. 13–24, 2010.

[34] J. M. Rivero, J. Grigera, G. Rossi, E. R. Luna, and N. Koch, “Towards Agile Model-Driven Web Engineering *,” vol. 734, pp. 142–155, 2012.

[35] J. Zhang, B. Grove, J. Chung, and Y. Heights, “Mockup-driven Fast-prototyping Methodology for Web Application Development,” pp. 1–30.

[36] J. M. Rivero, S. Heil, J. Grigera, M. Gaedke, E. R. Luna, and G. Rossi, “An Extensible, Model-Driven and End-User Centric Approach for API Building” in Proceedings of the 14th International Conference on Web Engineering, 2014, pp. 494-497.”

[37] J. M. Rivero, S. Heil, J. Grigera, M. Gaedke, and G. Rossi, “MockAPI: An Agile Approach Supporting API-first Web Application Development,” *Icwe 2013, Lncs 7977*, pp. 7–21, 2013.

[38] B. Mulloy, “Crafting Interfaces that Developers Love.”

[39] “Logic-less templates.” [Online]. Available: http://mustache.github.io/.

[40] “JSON.” .

[41] 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,” pp. 1–40.

[42] “Web Ratio.” [Online]. Available: http://www.webratio.com.

[43] “WebRatio. Chosen by Innovators.” [Online]. Available: http://www.webratio.com/site/content/en/customers?link=oln266n.redirect&nav=page36.56.

[44] “IFML: The Interaction Flow Modeling Language.” [Online]. Available: IFML: The Interaction Flow Modeling Language .

23.1 ANEXO A. CÓDIGO DE LOS TAGS

El presente Anexo contiene fragmentos de código que se consideran relevantes para comprender el funcionamiento de los tags desarrollados.

1. ListTag

```
run: function(tag) {
      var self = this;
      self.getObjects(self, function(elements) {
      for (i in elements) {
            self.addElement(elements[i],self);
      }
});
```

2. PropertyListTag

```
run: function(domElement,objeto,tag) {
  var value="";
  if (objeto[this.getParameters().content]){
      value=objeto[this.getParameters().content];
  }
  if ((domElement.find(this.getParameters().selectorTag)).is('input'))
      domElement.find(this.getParameters().selectorTag).val(value);
  else
      domElement.find(this.getParameters().selectorTag).html(value);
  return objeto;
}
```

3. DeleteListTag

```
run: function(element,objeto,tag) {
  self=this;
  $(self.getParameters().selector)[self.getParameters().disparador](
  function (){
      var deleteObjects="";
      for (var i =0; i<tag.getSelectObjects().length; i++){
          deleteObjects=deleteObjects+tag.getSelectObjects()[i].id+",";
      }
      $.ajax({url:       self.getUrlPrefix()       +       "/"       +
      tag.getParameters().content+"/"+deleteObjects, type: "DELETE",
            success: function() {
              tag.run(tag);
            },error: function() {
              console.log("error");
            }
      })
  });
}
```

4. DataEntryTag

```
run: function(tag) {
  var JSONObject={};
  for (j in this.getSubTags()){
      JSONObject=this.getSubTags()[j].run(
                  $(this.getParameters().selector),JSONObject,this);
  }
  return JSONObject;
}
```

5. PropertyDataTag

```
run: function(domElement,objeto,tag) {
      var value="";
      if (objeto[this.getParameters().content]){
            value=objeto[this.getParameters().content];
      }
      if ($(this.getParameters().selector).is('input'))
            $(this.getParameters().selector).val(value);
      else
            $(this.getParameters().selector).html(value);
      return objeto;
}
```

6. SaveTag

```
run: function(element,object,tag) {
      self=this;
      $(self.getParameters().selector)["click"](function (){
      var JSONObject={};
      JSONObject=tag.createObject();
      $.ajax(
      {url: contentP, type: typeP, data:JSON.stringify(JSONObject),
            success: function() {
                  console.log("ok");
                  tag.cleanFields();
            },error:    function() {
                  console.log("error");
      }
      })
      return object;
}
```

7. DeleteDataTag

```
run: function(element, objeto,tag) {
      self=this;
      element.find(self.getParameters().selector)
      [self.getParameters().disparador]
      (function (){
            $.ajax({url:      self.getUrlPrefix()      +      "/"      +
            tag.getParameters().content+"/"+objeto.id, type: "DELETE",
            success: function() {
                  console.log("ok");
                  tag.run(tag);
            },error:function() {
                  console.log("error");
            }
      })
}
```

8. LinkTag

```
run: function(element,object,tag) {
      var self = this;
      element.find(self.getParameters().selector.replace(
      tag.getParameters().selector,"")).click(function(e) {
      window.location =
      self.getParameters()["page"]+".html"+window.location.search;})
}
```

9. TransferTag

```
run: function(element,object,tag) {
      var self=this;
      element.find(self.getParameters().selector).click(function(e) {
            object[”_id”]=self.getObjectId();
            self._saveObject(object);
 })
}
```

10. DataTag

```
run: function(tag) {
      self=this;
      self._getObject(self.getObjectId(),function(JSONObject){
        for (j in self.getSubTags()){
            self.getSubTags()[j].run($(self.getParameters().selector),
            JSONObject,self);
        }
 })
}
```

11. DataNavTag

```
run: function(domElement,objeto,tag) {
      var value=””;
      self=this;
      self.getObject(self, objeto, function(objectNav) {
            objectNav=JSON.parse(objectNav);
            for (j in self.getSubTags()){
                  self.getSubTags()[j].run(domElement,objectNav,self);
            }
      })
}
```

12. SelectTag

```
run: function(element, objeto,tag) {
      self=this;
      element.find(self.parameters.selectorTag)[”change”](
      function (){
            if(this.checked ==true)
                  tag.setSelectObject(objeto);
            else
                  tag.removeSelectObject(objeto);
             }
 )}
})
```

24.1__