



TESINA DE LICENCIATURA

Título: Detección y resolución de conflictos en requerimientos utilizando mockups

Autores: Diaz Cavuoti Gonzalo Emanuel

Director: Matías Urbietta

Codirector: José Matías Rivero

Carrera: Licenciatura en Sistemas

Resumen

La elicitación de requerimientos implica entender las necesidades del usuario, incluso cuando las reglas de negocios son desconocidas o varían durante el desarrollo de software. Por lo que si se produce un error durante la especificación, la reparación o resolución del mismo va a ser costosa de resolver para los analistas del proyecto. Actualmente es muy frecuente el uso de metodologías ágiles durante el desarrollo, haciendo empleo de la construcción de mockups que representen las funcionalidades a implementar. Teniendo esto en cuenta, en este trabajo se propone un enfoque que tiene como objetivo enriquecer los mockups con anotaciones a partir de una gramática de usuario final. Para eso se desarrolló una herramienta que permite etiquetar widgets a partir de un catálogo de etiquetas destinado al usuario, para obtener una descripción formal con el fin de que cualquier pieza esté bien descrita y sin ambigüedades. Por otro lado la herramienta posibilita detectar conflictos o inconsistencias entre mockups, donde los mismos modelen el mismo requerimiento pero que pertenecen a historias de usuario diferente, permitiendo tener un control más exhaustivo de los cambios en los requerimientos.

Palabras Claves

- *Mockups.*
- *Especificación de requerimientos.*
- *Catálogo de anotaciones.*
- *Etiquetado de widgets.*
- *Descripción formal de mockups.*
- *Metodologías Ágiles*

Trabajos Realizados

En primer lugar se realizó una investigación bibliográfica de los temas a desarrollar. Luego se definió un catálogo con las anotaciones que van a ser utilizadas para etiquetar mockups. Se programó una herramienta que realiza el etiquetado de los mockups, donde una vez etiquetados permite detectar conflictos e inconsistencias entre diferentes versiones de los mismos. En caso de existir conflictos, la herramienta ofrece posibles refactoring o soluciones dependiendo la naturaleza del mismo. Se realizaron pruebas con 26 analistas donde describieron mockups etiquetados y sin etiquetar, y se analizó la eficiencia de la herramienta en comparación a los datos obtenidos de las pruebas.

Conclusiones

Después de implementada la herramienta y en función del análisis de los resultados obtenidos con los analistas, puede observarse que el uso de la herramienta mejora tanto la especificación de los requerimientos como la eficiencia en la detección de diferencias entre mockups. Si bien existe una tendencia a la mejora y reducción de errores, se necesita evaluar la herramienta con un número mayor de casos para confirmar dicha tendencia.

Trabajos Futuros

Sería interesante ampliar el catálogo de anotaciones disponibles a la hora de etiquetar mockups, donde dicha ampliación esté relacionada con la similitud de las características de los requerimientos. Donde existan etiquetas destinadas a un entorno específico como ser sistemas contables, administrativos, etc. A su vez se puede pensar en la posibilidad de extender los tipos de conflictos o inconsistencia que detecta la herramienta. Mencionadas sugerencias requieren un análisis profundo del dominio al cual se quiera estudiar.

Índice general

Índice de figuras

1. Introducción	1
2. Objetivo	5
3. Trabajo Relacionado	6
4. Antecedentes	9
4.1. Tecnologías utilizadas	12
4.1.1. JAVA	12
4.1.2. Spring Framework	13
4.1.3. Drools	14
4.1.4. PEGJs	15
4.1.5. ANTLR	16
4.1.6. MongoDB	17
4.1.7. HTML5, JavaScript, jQuery, AJAX, CSS	18
5. Catálogo de anotaciones de gramática del usuario final	21
5.1. Implementación de la gramática usuario final	25
5.1.1. Gramática PEGJS	26
5.1.2. Gramática ANTLR	28
6. Caracterización de conflictos de requerimientos en Aplicaciones Web	29

7. Detección y resolución de conflictos	31
7.1. Modelado y simulaciones de historiales de usuario (pasos 1 y 2)	33
7.2. Mejoramiento de los requerimientos (paso 3)	34
7.2.1. Descripción de datos coloquiales	35
7.3. Detección de diferencias (paso 4) y conciliación (paso 5)	36
7.3.1. Tipos de conflictos a detectar	38
7.3.2. Tipos de advertencias a detectar	53
7.4. Evolución de requerimientos de gestión	60
8. Arquitectura de la herramienta	62
9. Evaluación de la herramienta	65
9.1. Objetivos, hipótesis y variables	65
9.2. Métricas y materiales	66
9.3. Asignaturas, Instrumentación y Recolección de Datos	67
9.4. Análisis	68
9.5. Evaluación de Resultados e Implicación	69
9.6. Amenazas a la validez	71
10. Conclusiones y Trabajos Futuros	73
Bibliografía	75

Índice de figuras

1.1. Mockup de la versión web de una aplicación de comercio electrónico en la sección de teléfonos inteligentes.	3
4.1. Diagrama de clases de la herramienta	10
5.1. Mockup descripto utilizando gramática usuario final	25
7.1. El proceso general para detectar conflictos de requerimientos	31
7.2. Mockup descripto con gramática de usuario final utilizando la herramienta	36
7.3. Mockup A conflicto diferente tipo de dato	39
7.4. Mockup B conflicto diferente tipo de dato	39
7.5. Resultado de detección de conflictos entre mockup A y B diferente tipo de dato	40
7.6. Mockup A conflicto de navegación	42
7.7. Mockup B conflicto de navegación	42
7.8. Resultado de detección de conflictos entre mockup A y B conflicto de navegación	43
7.9. Mockup A conflicto mismo atributo distinta entidad	45
7.10. Mockup B conflicto mismo atributo distinta entidad	45
7.11. Resultado de detección de conflictos entre mockup A y B conflicto mismo atributo distinta entidad	46
7.12. Mockup A conflicto mismo valor diferente tipo de widget	48
7.13. Mockup B conflicto mismo valor diferente tipo de widget	48

7.14. Resultado de detección de conflictos entre mockup A y B conflicto mismo valor diferente tipo de widget	49
7.15. Mockup A conflicto mismo atributo diferente parámetros	51
7.16. Mockup B conflicto mismo atributo diferente parámetros	51
7.17. Resultado de detección de conflictos entre mockup A y B conflicto mismo atributo diferente parámetros	52
7.18. Mockup A warning mismo anotación diferente tipo de widget	54
7.19. Mockup B warning mismo anotación diferente tipo de widget	55
7.20. Resultado de detección de conflictos entre mockup A y B warning mismo anotación diferente tipo de widget	55
7.21. Mockup A warning diferentes botones mismo destino	58
7.22. Mockup B warning diferentes botones mismo destino	58
7.23. Resultado de detección de conflictos entre mockup A y B warning diferentes botones mismo destino	58
7.24. Impacto de los requerimientos en el modelo de la aplicación	61
8.1. Representación de las tecnologías utilizadas	63
8.2. Arquitectura de la herramienta desarrollada	64
9.1. Resultados de la muestra	70

1. Introducción

La elicitación de requerimientos de una aplicación implica entender las necesidades de las partes interesadas, incluso en los casos en que las reglas de negocio pueden ser parcial o totalmente desconocidas para los analistas que realizan la elicitación. A menudo, los requerimientos son acordados por las partes interesadas de tal manera que la semántica y los significados de cada término sean bien comprendidos. Sin embargo, cuando existen diferentes puntos de vista [10] del mismo concepto de negocio, pueden surgir ambigüedades y/o incoherencias que sean perjudiciales para la Especificación de Requerimientos de Software (SRS). Tradicionalmente, las tareas de conciliación se realizan utilizando herramientas basadas en reuniones [12], con el fin de eliminar requerimientos ambiguos e inconsistencias. Cuando las inconsistencias no son detectadas a tiempo (siendo esta una de las razones más severas de superar el costo del proyecto [11], [30]), estas pueden convertirse en defectos en la aplicación. En este contexto, el esfuerzo para corregir fallas es de mayor magnitud que la corrección de requerimientos en las primeras etapas [10, 2, 20].

Durante los últimos años, las metodologías ágiles han sido ampliamente adoptadas y su uso se ha convertido en un factor clave en el éxito de un proyecto. En la práctica, estas metodologías reduce la brecha entre las expectativas y los resultados, por tener ciclos cortos de desarrollo en los que se entrega un conjunto de funcionalidades que se validan contra los requerimientos al final de cada iteración. Esta práctica a menudo no se centra en documentar la solución (diagramas de clase, diagramas de implementación, etc) como se hace, por ejemplo, en aproximaciones de cascada o RUP.

Una de las herramientas más importantes adoptada para documentar por los profesionales es la confección de maquetas de interfaz de usuario. Mediante el uso de esta técnica, la forma en que los escenarios empresariales se instancian dependen de las

descripciones textuales de las historias de usuario y diseños de página de baja o alta fidelidad que comunican fácilmente el comportamiento de las aplicaciones a las partes interesadas. Las historias de usuario (en inglés *user story*) son una representación de un requerimiento escrito en una o dos frases utilizando el lenguaje común del usuario. Son utilizadas habitualmente en metodologías de desarrollo ágiles para la especificación de requerimientos; dado que permite una forma de administración rápida de estos, sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para su elaboración. Permite responder rápidamente a los requerimientos cambiantes.

Por otro lado, las maquetas de interfaz de usuario (en inglés *mockup*) son un prototipo de baja/media fidelidad de una página web o pantalla de una aplicación; los mockups se utilizan para identificar los elementos que se mostrarán en la página o pantalla, tales como: navegación, secciones de contenido, imágenes y/o necesidades de medios, elementos de formulario y llamadas a la acción (CTA). Existen desde los más básicos, hasta los más avanzados que pueden llegar a reproducir el diseño de pantalla completa. Generalmente, las personas que los crean, participan en las reuniones que se realizan con el cliente. Las herramientas de mockups son útiles para describir escenarios en los que la información ejemplificativa está lo más cerca posible de un uso real antes del desarrollo del sistema.

Sin embargo, esta información es informal y permite confusiones para las diferentes partes involucradas.

Por ejemplo, podríamos considerar una aplicación de ventas, que muestra una lista de productos (Figura 1.1). Para cada producto tenemos una imagen, un precio, un título, una descripción, una cantidad vendida, una ubicación, un icono para indicar si el producto es nuevo, otro para indicar si el vendedor es premium y uno que indica si el producto tiene el envío libre o no. Aunque esta información es ilustrativa, carece de la precisión necesaria para describir formalmente los requerimientos para el desarrollo del software. El lector del mockup no es capaz de distinguir si el precio del producto, es el precio regular o el precio de Internet, en realidad se muestra la etiqueta \$499, o si la etiqueta que muestra el recuento de los productos vendidos está relacionada con el



Figura 1.1: Mockup de la versión web de una aplicación de comercio electrónico en la sección de teléfonos inteligentes.

producto o a otro concepto de negocio, tales artículos vendidos por el mismo vendedor. Además la secuencia de navegación de los botones no está descripta. Estos aspectos pueden ser pasados por alto fácilmente tanto por los clientes como analistas pero pueden tener consecuencias negativas para el desarrollo de software. Los desarrolladores pueden definir por sí mismos la información faltante basada en esta información imprecisa donde cualquier error se detectará posteriormente con una resolución costosa. Esto se debe al hecho de que la herramienta de maquetado no es una especificación formal y no proporciona la posibilidad de enumerar abstracciones tales como variables y entidades como hace UML.

Para empeorar la situación, siempre que se planifiquen nuevos requerimientos en la iteración de las historias de usuario, uno o más conflictos pueden aumentar. Las inconsistencias también pueden surgir de nuevos requerimientos, que introducen

nuevas funcionalidades o mejoras a la aplicación o incluso de los requerimientos existentes cambien durante el proceso de desarrollo. Supongamos que para el caso de uso expuesto en la Figura 1.1, hay una nueva interfaz de usuario ligeramente diferente basada en móviles. Este nuevo mockup se utiliza para describir una nueva iniciativa comercial y tiene diferentes reglas de negocio que no se puede apreciar solo viendo el mockup, por ejemplo descuentos promocionales o beneficios de envío gratuito. Este tipo de inconsistencias ciertamente no son detectadas porque falta una documentación adecuada. Durante el proceso de desarrollo, en la mayoría de los casos, los desarrolladores asumen que el código es una fuente más confiable que otros documentos [27]; especialmente porque los requerimientos, diseños y modelos no pueden mantenerse actualizados cuando el código cambia. En este contexto, es difícil rastrear las definiciones de los requerimientos (y sus cambios) y el conocimiento de la aplicación que se está desarrollando se convierte en tácito. Por lo tanto, un miembro del equipo que quiere entender alguna característica de la aplicación que se basa en mockups simples e historias de usuarios, debe pedir a sus compañeros información tácita. Esta situación introduce algunos riesgos: información parcial, información engañosa o incluso información errónea.

En el capítulo 2 se van a plantear los objetivos a realizar en el trabajo. En el capítulo 3 se hacen mención a trabajos relacionados con la problemática planteada. En el capítulo 4 se presentan los antecedentes en que se basa el enfoque propuesto y se describen las tecnologías utilizadas para llevarlo a cabo. En el capítulo 5 se define y describe la gramática de usuario final para describir formalmente los mockups. En el capítulo 6 se hace una caracterización de los conflictos en aplicaciones web. En el capítulo 7 se presenta la herramienta que brinda apoyo para el análisis de detección de conflictos. En el capítulo 8 se describe la arquitectura de la herramienta desarrollada. En el capítulo 9 se hace una evaluación del enfoque propuesto en este trabajo.

2. Objetivo

Para hacer frente a la problemática planteada, se propone como propósito alcanzar los siguientes objetivos:

- Una notación coloquial y amistosa para describir datos, navegación, requerimientos de negocios e interacción sobre especificaciones de mockups, basados en MockupDD [20].
- Un proceso en el que podemos hacer un análisis de consistencia más allá de las diferencias visuales, lo que ayuda a descubrir, clasificar y, si es necesario, refactorizar inconsistencias de modelos.
- Una herramienta web que apoye la mejora mencionada anteriormente.
- Una evaluación de la contribución de este enfoque.

Se decidió utilizar mockups como artefactos de requerimientos básicos para asistir en la detección de conflictos de requerimientos semiautomáticos dado que (1) se usan ampliamente en metodologías ágiles, que actualmente son la tendencia en la industria y (2) su nivel de detalle (en los digitales) es suficiente para procesarlos automáticamente y detectar inconsistencias con mínima participación humana - sólo relacionada con la desambiguación. La intención principal del trabajo, es plantear un enfoque en donde se pueda facilitar la especificación formal de widgets que contiene un mockup.

3. Trabajo Relacionado

El análisis y detección de conflictos, errores y equivocaciones en la fase de requerimientos es una de las tareas más críticas en la ingeniería de requerimientos [25].

Hay varios enfoques para el tratamiento de los requerimientos, una visión global presentada en [7] divide esta fase en tres tareas principales: captura de requerimientos, definición de requerimientos y validación de requerimientos. En [7], los autores analizaron la forma en que los enfoques de ingeniería Web abordan estas tres fases y concluyen que la mayoría de los enfoques utilizan técnicas clásicas para hacer frente a los requerimientos. De acuerdo con estas técnicas, existen cuatro técnicas principales para la validación de requerimientos: revisiones, auditorías, matriz de trazabilidad y prototipos. En la literatura de ingeniería Web, la validación de requerimientos es uno de los temas menos tratados. Además, ninguna de estas técnicas ofrece una detección sistemática de conflictos en los requerimientos.

Las herramientas de mockups están aumentando la atención en el campo de la ingeniería de requerimientos ya que ayudan a construir las especificaciones de la interfaz de usuario junto con los usuarios finales y también a descubrir y definir los requerimientos que no son de interfaz de usuario en un lenguaje que esté más cerca de ellos, opuesto a especificaciones textuales sencillas [16, 19]. También se ha demostrado que las maquetaciones son un método efectivo para capturar los requerimientos fluido [24]. Aquellos que generalmente se expresan oralmente o informalmente y una parte implícita (y usualmente pérdida) del proceso de elicitación. El uso de prototipos de interfaces de usuario con estructura estática para definir modelos conceptuales ya se ha mostrado en [18]. El proceso ICONIX [22] propone iniciar con prototipos de interfaz gráfica de usuario (GUI) como un primer artefacto de requerimientos. Si bien esto

puede proporcionar algunas pautas iniciales, en ese trabajo los autores no proporcionan ningún lenguaje o guía formal para definir los requerimientos de datos. El enfoque de simulacro de datos propone utilizar mockups con anotaciones formales para especificar y construir modelos conceptuales (en el que se basa el lenguaje propuesto en este documento) utilizando una herramienta específica. Sin embargo, ese trabajo se limita a la especificación del modelo conceptual utilizando UML y solo se mide la eficiencia de la definición del modelo. En este trabajo, se propone un enfoque que está vinculado a una herramienta específica y se centra en mejorar la recopilación de requerimientos mediante el uso de anotaciones fáciles de usar.

Los últimos años, se han realizado investigaciones sobre diferentes estrategias para capturar requerimientos de software web [8, 28]. Específicamente, se pueden mencionar WebSpec [13], un lenguaje específico para capturar interacción y requerimientos de navegación en aplicaciones web. Similar a otros enfoques como Molic [17] y WebRE [8], que proporcionan primitivas para describir entradas, salidas, navegaciones, transiciones de interfaz, etc; ayudando a describir las principales preocupaciones de aplicación de una manera más precisa. Estos enfoques fueron diseñados para ser conectados en enfoques dirigidos por modelo y no encajan fácilmente en los procesos de desarrollo ágil.

Otra tendencia general es el uso del proceso orientado a objetivos. En este mecanismo, durante las primeras fases de los proyectos, se definen los objetivos del sistema. Se especifica un conjunto de requerimientos iniciales para cubrir estos objetivos y posteriormente, en cada iteración, estos se estudian y describen con más detalle y se pueden dividir en sub-requerimientos que están más cerca del código. Con este proceso, se detectan inconsistencias que miden la trazabilidad entre los objetivos y los requerimientos. Esta idea es seguida, por ejemplo por [15], en un enfoque basado en agentes.

Recientemente, algunos enfoques de diseño web, como WebML [4] y NDT [6], apoyan esta idea utilizando el paradigma impulsado por el modelo. Sin embargo, incluso ofreciendo soporte sistemático (o automático) para pruebas tempranas, la

detección de inconsistencias en la especificación de requerimientos continúa siendo mayormente trabajo manual y depende de la experiencia del analista y su capacidad para apoyar la revisión con clientes y usuarios. Centrándose solo en la detección de conflictos, en [3], se presenta un enfoque para detectar conflictos en preocupaciones. En este enfoque, los autores proponen el uso de un método de toma de decisiones de criterio múltiple para apoyar la gestión aspectual de conflictos en los requerimientos orientados a aspectos. La principal limitación de este enfoque es que se orienta a los aspectos del tratamiento de los requerimientos y sólo se ocupa de los conflictos de preocupación. En otras fases del ciclo de vida, el proceso de detección de conflictos ha sido intensamente investigado por la comunidad impulsada por modelos, centrada principalmente en conflictos de modelos UML. En [1] el autor propone detectar el conflicto en un doble proceso; analizar las diferencias sintácticas que plantean conflictos candidatos y comprender estas diferencias desde una visión semántica. En [26] un enfoque basado en el razonamiento de la lógica descriptiva; los modelos UML se transforman en documentos de lógica de descripción que posteriormente son procesados por un motor lógico de primer orden a cargo del razonamiento.

4. Antecedentes

El enfoque presentado en este documento se basa en gran medida en la metodología MockupDD [20], que es un enfoque de desarrollo impulsado por el modelo centrado en maquetaciones Interfaz de Usuario (UI). En el contexto del MockupDD, los mockups son los principales artefactos de especificación de requerimientos que, en lugar de descartarse como en los enfoques de desarrollo ágiles tradicionales, se reutilizan como base para definir especificaciones de software más complejas. Esta reutilización se logra a través de (1) la formalización de la estructura de los mockups y widgets a través de lo que se llama un modelo de interfaz de usuario estructural (SUI) y (2) la introducción de un conjunto de anotaciones formales sobre la estructura definida sobre dicho modelo [20]. Cada anotación colocada sobre el mockup formalizado representa una especificación independiente relacionada, por ejemplo, con el contenido, la navegación, el comportamiento o cualquier otro aspecto que se pueda especificar sobre una representación visual de la interfaz de usuario. La semántica definida para cada anotación permite, entre otras cosas, generar código o acciones interpretadas en tiempo de ejecución, traducirlas a representaciones semi-textuales para discutir los requerimientos capturados sobre las maquetaciones con las partes interesadas, etc. En este caso, las anotaciones se utilizan para formalizar y refinar los requerimientos para detectar conflictos. Por lo tanto, el enfoque depende de la definición de etiquetas formales que se definen en la Figura 4.1

Etiquetas (representadas por la clase Tag en el metamodelo) semánticamente se puede dividir entre:

- Etiqueta de datos: describe un tipo de objeto y/o propiedad del mismo, que tiene una representación gráfica en el mockup.

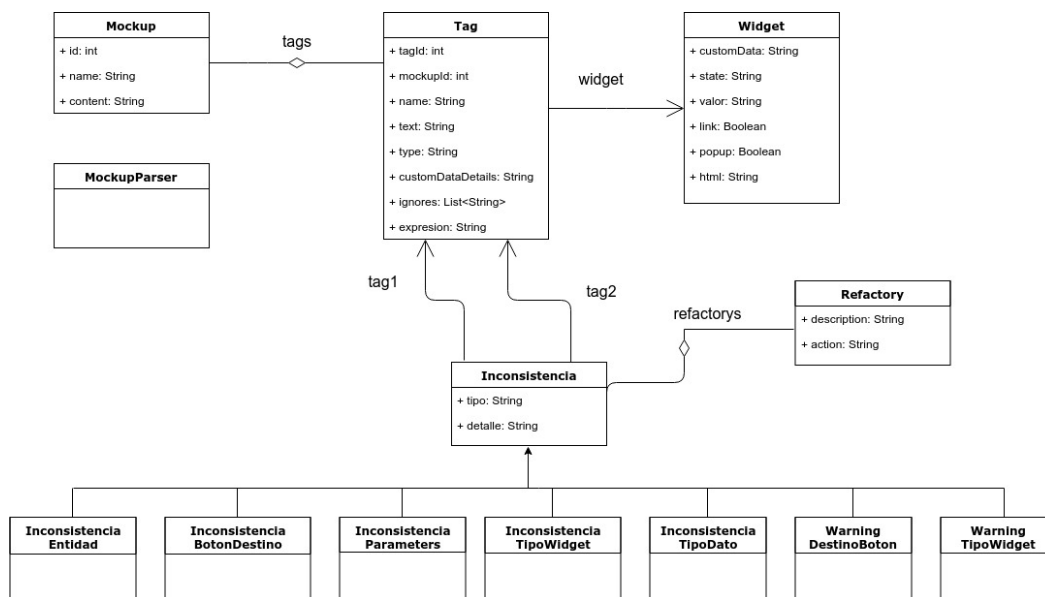


Figura 4.1: Diagrama de clases de la herramienta

- Activar la etiqueta: representa un objeto gráfico en el mockup que desencadena una activación de una interfaz de usuario diferente o parte de ella.
- Validar etiqueta: especifica una regla o criterio de validación que define si un elemento gráfico de entrada en el contenido del mockup es correcto o no.

En cuanto al comportamiento de la aplicación, se ha especificado un conjunto de etiquetas para describir como debe responder la aplicación después de que el usuario desencadene un evento relacionado con la manipulación de objetos declarados. Las etiquetas son:

- Guardar etiqueta / Borrar etiqueta: que representa que el elemento gráfico de interfaz de usuario asociado permite guardar o eliminar respectivamente una instancia de objeto de tipo de objeto representada por el parámetro de clase.
- Etiqueta de operación personalizada: representa una acción abstracta, descrita textualmente, que es activada por un elemento gráfico interactivo en el mockup. Se puede usar cuando la acción descrita no puede expresarse en términos de ninguna otra etiqueta.

El modelo de clases puede verse en la Figura 4.1. A continuación se hace una descripción de cada una de las clases para una mejor comprensión de su funcionamiento:

Mockup: Representa a una interfaz gráfica digitalizada, la cual es almacenada en base de datos como posibles mockups a ser utilizada por la herramienta para detectar conflictos y/o inconsistencias. Un mockup puede tener varios objetos de la clase Tag.

Tag: Se utiliza para representar la anotación/etiqueta que realiza el analista sobre el widget. En ella se almacena información tanto del widget como del significado semántico de dicha anotación. Esta información es utilizada por la herramienta para la sugerencia de posibles soluciones de conflicto. Un tag tiene un objeto de la clase Widget.

Widget: Representa los elementos que forman parte del mockup, el cual esta relacionado la etiqueta. Se almacena información de las propiedades que lo componen tales como el estado, el valor, código que lo define, etc. Esta información es utilizada por la herramienta para resolver conflictos y/o inconsistencias.

Inconsistencia: Se utiliza para representar el conflicto entre dos widgets que pertenecen a diferentes mockups. De la misma se guardan datos como el tipo y un detalle del motivo de la misma. La clase tiene 2 objetos de la clase Tag, que contienen las anotaciones para la cual la herramienta detectó conflicto. A su vez dependiendo del tipo de inconsistencia, existen varios objetos de la clase Refactory, que son las posibles soluciones que ofrece la herramienta. La clase Inconsistencia es superclase de todas las posibles inconsistencias o warning que se pueden detectar.

InconsistenciaTipoControl, InconsistenciaTipoDato, InconsistenciaBotonDestino, InconsistenciaEntidad, InconsistenciaParameters, WarningTipoControl, WarningDestinoBoton: Cada clase representa un tipo de conflicto/warning a detectar, por lo que el comportamiento de cada clase va a depender de la naturaleza del mismo. En cada subclase se implementa su lógica particular sobre los posibles refactoring y la implementación de cada uno.

Refactory: Representa una posible solución a la inconsistencia detectada. De cada refactory se guarda una descripción y las acciones a realizar en caso de aplicar dicha solución.

MockupParser: Representa la clase que permite a partir de una gramática explícita analítica reconocer un lenguaje formal en términos de un conjunto de reglas para reconocer cadenas de lenguaje. Dicha clase es generada a partir de una gramática definida para las etiquetas y se obtiene utilizando el framework ANTLR para interpretar lenguajes en JAVA, el cual va a ser descrito a continuación.

4.1. Tecnologías utilizadas

A continuación se presentarán y explicarán las tecnologías que se utilizaron y dieron soporte para el desarrollo de este trabajo. Las cuales facilitaron y posibilitaron llevar a cabo los propósitos del enfoque propuesto.

4.1.1. JAVA

Para el desarrollo del lado del servidor se utilizó el lenguaje de programación JAVA. En mi caso particular decidí este lenguaje por tener buena experiencia previa programando en JAVA, lo cual me llevó a elegirlo por sobre otros lenguajes.

Entre sus principales características se destacan que es un lenguaje sencillo de aprender, tiene una sintaxis muy similar a los lenguajes C y C++. Es orientado a objetos, lo cual permite utilizar todos los conceptos en los que se apoya esta técnica, tales como encapsulación, herencia, polimorfismo, abstracción, etc. El código generado por el compilador Java es independiente de la arquitectura que se disponga; por lo que podría ejecutarse en múltiples entornos y sistemas operativos. El motivo de esto es debido a quien realmente ejecuta el código generado por el compilador no es el procesador del ordenador directamente, sino que este se ejecuta mediante una máquina virtual; realizando una abstracción del hardware que se dispone. A su vez JAVA permite la inclusión de un amplio conjunto de bibliotecas o frameworks, lo cual

favorece el desarrollo de una aplicación dado que posibilita la reutilización de código y obtener soluciones a problemas los cuales ya fueron resueltos previamente por otros desarrolladores.

Otra característica del lenguaje, es que soporta multithreading, es decir un programa puede ser dividido en múltiples subprocesos, los cuales se ejecuten al mismo tiempo y realicen tareas diferentes comunicándose entre sí. Lo cual permite hacer un uso óptimo de los recursos disponibles, especialmente cuando se cuenta con más de una unidad de CPU.

En resumen, JAVA es un lenguaje robusto, legible, simple y potente. Es utilizado por una buena parte de la industria del software, por lo que existe una comunidad que brinda soporte a los problemas o inconvenientes con los que se puedan presentar.

4.1.2. Spring Framework

Spring es un framework de código abierto para el desarrollo de aplicaciones web, escrito en JAVA, que se basa en el patrón de arquitectura MVC (Modelo - Vista - Controlador), donde se busca la separación de los datos, con la lógica de negocios y las vistas. Entre los cuales incluye los siguientes módulos:

- Contenedor de inversión de control: permite la configuración de los componentes de aplicación y la administración del ciclo de vida de los objetos Java, se lleva a cabo principalmente a través de la inyección de dependencias.
- Acceso a datos: se trabaja con RDBMS en la plataforma JAVA, usando Java Database Connectivity y herramientas de ORM (Mapeo objeto relacional) con bases de datos NoSQL.
- Modelo vista controlador: Un framework basado en HTTP y servlets, que provee herramientas para la extensión y personalización de aplicaciones web y servicios web REST.

- Gestión de transacciones: unifica distintas APIs de gestión y coordina las transacciones para los objetos JAVA.
- Testing: Soporte de clases para desarrollo de unidades de prueba e integración.

Personalmente elegí Spring, dado que es un framework el cual esta basado en MVC, lo que permite desarrollar aplicaciones web sobre este paradigma, y la posibilidad de tener un ORM para realizar una programación orientada a objetos con un sencillo acceso a una base de datos. Después de haber participado en diferentes proyectos, tanto en la facultad como laborales habiendo utilizado tanto el concepto de MVC y ORM, considere que utilizar dichas características para el desarrollo de la herramienta era una opción acertada; dado que permite separar los datos, de la lógica de negocios y de la forma en que se muestran esos datos. A su vez, este framework es uno de los más utilizados dentro de JAVA ¹ para aplicaciones web por lo que existe en la Internet documentación y soporte suficiente sobre su uso y posibles problemas que puedan ocurrir.

4.1.3. Drools

Drools es un sistema de gestión de reglas de negocio (BRMS, por las siglas en inglés de Business Rule Management System) open source, que puede usarse dentro de una aplicación JAVA. Su principal funcionalidad surge ante la necesidad de centralizar y gestionar la lógica de negocio. Para ello, se codifica dicha lógica en forma de reglas, que sirven para tomar decisiones dentro de un contexto. Estas reglas se ejecutan dentro de un motor de reglas (BRM). Las reglas, por lo tanto, no sirven únicamente para representar la lógica de negocio, sino también para ejecutarlas. Una regla de negocio es una sentencia del tipo cuando/entonces (when/then). Cuando se cumple una condición, se desencadena una acción determinada. Sus principales características son:

- Motor de reglas: basado en el algoritmo Rete que permite que las reglas se ejecuten de manera muy eficiente. Dicho algoritmo se denomina “Drools Expert”.

¹<https://openwebinars.net/blog/los-7-mejores-frameworks-de-java-de-2017/>

- Lenguaje DRL: que permite la creación de reglas de manera sencilla. Permite también la creación de reglas en lenguaje específico usando DSL.
- BRMS (Guvnor): permite gestionar las reglas de manera centralizada con una interfaz web.

El uso de Drools dentro de la herramienta es un componente de la arquitectura muy importante, dado que su función primordial es la de toma de decisiones, y en donde se declaran las reglas que determinan si existe o no conflicto entre elementos del mockup a partir del modelo de datos. Opte por utilizar Drools dado que permite separar la lógica de negocios dentro de la aplicación, y posibilita realizar un mejor diseño de la solución; en el caso particular de la herramienta, esto trae el beneficio de que si se presentan cambios en las condiciones en las que se detectan conflictos, dichos cambios solo repercuten en el motor de reglas. Además teniendo la ventaja que tiene compatibilidad con JAVA, el lenguaje de programación utilizado para el desarrollo de la herramienta.

4.1.4. PEGJs

PEGJs es una librería JavaScript que genera un analizador a partir de una gramática que describe la entrada esperada y puede especificar que devuelve el analizador (utilizando acciones semánticas en partes coincidentes de la entrada). El analizador en si mismo, es un objeto de JavaScript con una API simple. La gramática consiste en un conjunto de reglas. Cada regla tiene un nombre que la identifica, y una expresión de análisis sintáctico (por ejemplo `digits:[0-9]+ return parseInt(digits.join(), 10);`) que define un patrón para hacer coincidir con el texto de entrada y posiblemente contenga algún código JavaScript que determina qué sucede cuando el patrón coincide con éxito.

Sus principales características son:

- Sintaxis gramatical simple y expresiva.
- Integra análisis léxico y sintáctico.

- Se puede usar desde su navegador, desde la línea de comando o mediante la API de JavaScript.
- Funciona en todos los navegadores actuales (Internet Explorer 8+, Edge, Firefox, Chrome, Safari, Opera)

Su empleo dentro de la herramienta, permite analizar y verificar las anotaciones que se realizan sobre los widgets del mockup desde el editor web. A partir de dichas anotaciones, se define una gramática la cual determina las reglas que deben cumplir una anotación, permitiendo instanciar una estructura de datos en JavaScript en caso de ser una anotación válida. Esta estructura, se utiliza para guardar y relacionar las anotaciones con los widgets a la cual está asociado mediante la invocación AJAX de una API REST. Para poder lograr esto, se hace uso de una librería jQuery, dichos conceptos van a ser descriptos posteriormente.

Particularmente elegí utilizar PEGJs, dado que es una librería que permite definir una gramática de una manera muy simple, teniendo la ventaja de poder utilizar una versión online donde se pueden definir las reglas que componen la gramática y al mismo tiempo ir testeando la misma con un parámetro de entrada. Permitiendo observar en tiempo real los datos de salida que componen el parser.

4.1.5. ANTLR

ANTLR es un potente generador de analizadores para leer, procesar, ejecutar o traducir texto estructurado o archivos binarios, el cual está escrito en JAVA. Es ampliamente utilizado para construir idiomas, herramientas y frameworks. A partir de una gramática, ANTLR genera un analizador que puede construir árboles de análisis y crea una clase de análisis que hace que sea fácil responder al reconocimiento de datos a partir de una entrada. Dicha clase se llama MockupParser y se encuentra modelada en el diagrama de UML (ver 4), la cual permite analizar las anotaciones en base a las reglas definida en la gramática. A diferencia de PEGJs, ANTLR se encuentra del lado del servidor, lo que posibilita determinar las condiciones que deben cumplirse para decidir

si existe o no conflicto, a partir de los datos almacenados de los mockups. Es decir, que podemos comprender a partir de la semántica adquirida desde la gramática, el significado que el analista quiso darle a la anotación, y extraer la información necesaria para ejecutar las condiciones definidas en el motor de reglas de Drools. El catálogo de anotaciones de gramática del usuario final va a ser presentando en la sección 5

Personalmente elegí usar ANTLR ya que es una de las librerías más conocidas para crear gramáticas, teniendo como particularidad que tiene soporte para JAVA, siendo el lenguaje de programación utilizado para el desarrollo. Cuenta con abundante documentación en la red, y al ser una librería open source cuenta con soporte por parte de otros desarrolladores.

4.1.6. MongoDB

MongoDB es una base de datos NoSQL y permite implementar aplicaciones más ágiles y escalables. En lugar de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos similares a JSON con un esquema dinámico (MongoDB utiliza una especificación llamada BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. Utiliza objetos JSON para guardar y transmitir la información. JSON es un estandar web hoy en día, por lo que es una gran ventaja que tanto la aplicación web y la base de datos tengan el mismo formato de la información.

Soporta la búsqueda por campos, consultas de rangos y expresiones regulares. Las consultas pueden devolver un campo específico del documento pero también puede ser una función JavaScript definida por el usuario. Cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer índices secundarios. El concepto de índices en MongoDB es similar a los encontrados en base de datos relacionales. Soporta el tipo de replicación primario-secundario. Cada grupo de primario y sus secundarios se denomina replica set. El primario puede ejecutar comandos de lectura y escritura. Los secundarios replican los datos del primario y solo se pueden usar para lectura o para copia de seguridad, pero no se pueden realizar escrituras. Los

secundarios tiene la habilidad de poder elegir un nuevo primario en caso de que el primario actual deje de responder.

Con MongoDB se puede escalar una arquitectura de forma horizontal usando el concepto de “shard”. El desarrollador puede elegir una clave de sharding, la cual determina como serán distribuidos los datos de una colección. Los datos son divididos en rangos (basado en la clave de sharding) y distribuidos a través de múltiples shard. Cada shard puede ser una replica set. MongoDB tiene la capacidad de ejecutarse en múltiple servidores, balanceando la carga y/o replicando los datos para poder mantener el sistema funcionando en caso que exista un fallo de hardware.

Particularmente opte por utilizar MongoDB, ya que es una base de datos NoSQL que se adapta al propósito que requería, ya que esta orientada a documentos, donde un documento es capaz de almacenar toda la información necesaria que lo define, aceptando todo tipo de datos (incluidos arrays y otros subdocumentos). A su vez me permitió adaptar el esquema de base de datos a las necesidades de la aplicación rápidamente, disminuyendo tiempo de definición de esquema. Esto es así porque permite modificar el esquema desde el código de la aplicación, sin tener que realizar labores de administración de la base de datos como ocurre con las bases de datos relacionales.

4.1.7. HTML5, JavaScript, jQuery, AJAX, CSS

Las tecnologías utilizadas del lado del cliente, son largamente conocidas en el desarrollo de aplicaciones web, sin embargo se va a hacer una de breve descripción de las mismas. HTML5 es la última versión de Hypertext Markup Language, el código estándar que es utilizado para crear páginas web y puede ser interpretado por cualquier navegador. Esta nueva versión, y en conjunto con CSS3, define los nuevos estándares de desarrollo web, rediseñando el código para resolver problemas y adaptarse así a nuevas necesidades. Se agregan nuevas etiquetas y atributos para multimedia que permiten reproducir audios y videos. Al ser un lenguaje estándar es soportado por cualquier navegador web, lo que permite ser utilizado desde cualquier dispositivo ya sea notebook, tablet, celular, etc.

JavaScript es un lenguaje de programación interpretado, que se define como orientado a objetos basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque existe una forma de JavaScript del lado del servidor. Para simplificar el uso del lenguaje hice uso de la librería jQuery basada en JavaScript, que permite facilitar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Dichas características simplificaron el desarrollo del editor web, ofreciendo todos los instrumentos necesarios para alcanzar todas las acciones realizadas en el navegador. Es software libre y de código abierto, siendo una de las utilizadas ² en JavaScript y compatible con la mayoría de los navegadores (Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ y Google Chrome 8+.5).

AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones. AJAX es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página.

Su utilización se efectúa a través de la librería jQuery, permitiendo una comunicación entre el cliente y el servidor; en el caso específico de la herramienta, entre el editor web de mockups y la API REST que se ejecuta en el servidor.

CSS es un lenguaje de diseño gráfico para definir y crear la presentación de un documento HTML. Abarca cuestiones relativas a fuentes, colores, márgenes, líneas, altura, anchura, imágenes de fondo, posicionamiento avanzado, etc. A través de dicho lenguaje, se especifica el estilo que puede adoptar los elementos que forman una página

² «Usage of JavaScript libraries for websites». http://w3techs.com/technologies/overview/javascript_library/all

web, es decir determinar la forma en que pueden ser visualizados. CSS3 es la última evolución del lenguaje que permite realizar animaciones dentro de las páginas web, entre otras novedades.

5. Catálogo de anotaciones de gramática del usuario final

En esta sección, se presentará el catálogo de anotaciones de usuario final para enriquecer las especificaciones del mockup que se utilizarán en la herramienta. Con el fin de mejorar la descripción de los elementos del mockup para resolver la falta de formalidad, en este paso se utilizara un lenguaje específico de dominio denominado gramática de usuario final (EUG) [21] que se centra en describir la fuente de información, el formato y las relaciones de información. Cada anotación es una definición coloquial estructurada que es legible por los usuarios finales porque no presenta cualquier concepto técnico que pudiera limitar su comprensión. A continuación, se introducen patrones de anotaciones y su descripción.

- “Mockup Name” view (ID)
 - Define un ID (número) para un mockup que haga referencia a un destino para la navegación / activación por otra etiqueta.
- a[n] [list of] class
 - Indica que un objeto o una lista de objetos de clase se muestra o puede manipularse en la interfaz de usuario. Por ejemplo, una lista en un mockup que muestra un índice de productos.
- Class’s attribute [which is a datatype | with options: value1 , . . . , and valueN]. [It’s required | It matches regularExpresion]
 - Especifica que el atributo de un objeto de clase es mostrado o puede ser editado a través de un widget gráfico subyacente. Opcionalmente, se puede

definir un tipo de datos para ese atributo (una fecha, cadena, entero, decimal, booleano, entero en enumerativo o Blob). Si no se especifica ningún tipo de datos, se asume una cadena. En el caso de un enumerativo es posible listar posibles valores utilizando la cláusula “with options: o1, o2, ..., oN”. Al final de la definición del atributo, se tiene la posibilidad de agregar un parámetro a la anotación indicando si el atributo es requerido o agregar una expresión regular con la cual se debe validar el formato.

- Class1 has a [n] [optional] [list of] Class2, called aRealName
 - Indica que un objeto de Class2 se muestra o puede ser manipulado a través del elemento subyacente en la interfaz de usuario. Sin embargo, este elemento se obtiene navegando desde una asociación denominada aRealName desde otro elemento de la clase Class1.
- Subclass is a type of Superclass
 - Indica que un objeto de clase Subclass se muestra o puede manipularse en la interfaz de usuario y que la clase de este objeto (Subclass) hereda de otra llamada Superclass.
- Navigates to destination / Opens a popup destination
 - Indica que al ejecutar una acción predeterminada sobre el elemento gráfico subyacente (por ejemplo, un clic), el mockup de destino se mostrará, se navegará o se enfocará. El mockup de destino debe ser etiquetado con: “Mockup Name” view (ID).
- Class’s attribute is required
 - Indica que se requiere un valor no vacío para el atributo de la clase.
- Class’s attribute min value is valorMínimo

- Indica que los valores para el atributo de la clase debe ser mayor a valorMínimo.
- Class's attribute max value is ValorMáximo
 - Indica que los valores para el atributo en clase debe ser menor a ValorMáximo.
- Class's attribute values must be between valorMínimo and ValorMáximo
 - Indica que los valores para el atributo en clase debe ser menor o igual que un valor máximo y/o mayor o igual que un valor mínimo.
- Class's attribute matches regularExpresion
 - Se podría requerir que la información se formatee para que coincida con un patrón (expresión regular). Por ejemplo, las fechas, números de teléfono y datos de códigos tienen restricciones de formato específicas. Indica que el atributo que pertenece a la clase debe cumplir el patrón regularExpresion.
- Saves a Class
 - Indica que al ejecutar una acción predeterminada sobre el elemento gráfico subyacente (por ejemplo, un clic), se creará una instancia de la clase (que se está editando).
- Deletes a Class
 - Indica que al ejecutar una acción predeterminada sobre el elemento gráfico subyacente, se borrará una instancia de la clase que se muestre o edite.
- Triggers "action description"
 - Indica que al ejecutar una acción predeterminada sobre el widget subyacente, se invocará una acción arbitraria (descrita textualmente). Esta construcción

se utiliza cuando el comportamiento esperado no está ya definido, pero se debe señalar.

Con las anotaciones descritas se consigue alcanzar una expresión más precisa sobre cada widget, permitiendo obtener a los analistas y programadores del proyecto mayor conocimiento sobre los mockups. Dicho conocimiento, posibilita a la herramienta semi-automatizar la detección de conflictos durante la especificación de requerimientos y otorga una fuente de información de los mismos. En la Figura 5.1 se muestra como se ve un mockup anotado con la gramática de usuario propuesta.

A partir del catálogo definido, se podrían definir múltiples versiones de la gramática de usuario final donde varíe el idioma en que se expresa. Esto posibilita reutilizar las anotaciones y aplicarlas en diferentes lenguas, donde solo se tendrían que escribir las reglas que forman parte de la nueva gramática. A continuación se van a mencionar algunos ejemplos del catálogo descrito anteriormente en Español:

- Registrar Usuario vista (1)
- El precio de Producto es de tipo Entero
- Empleado es un tipo de Persona
- El nombre de Persona es requerido
- Navega a Login vista (2)
- El valor mínimo de stock del Producto es 1
- Persona tiene una lista de Teléfono, llamada telefonos
- Guarda un Empleado
- El correo electrónico de Persona coincide con “Expresión Regular”
- La categoría del Producto con valores: Hogar, Computación y Salud

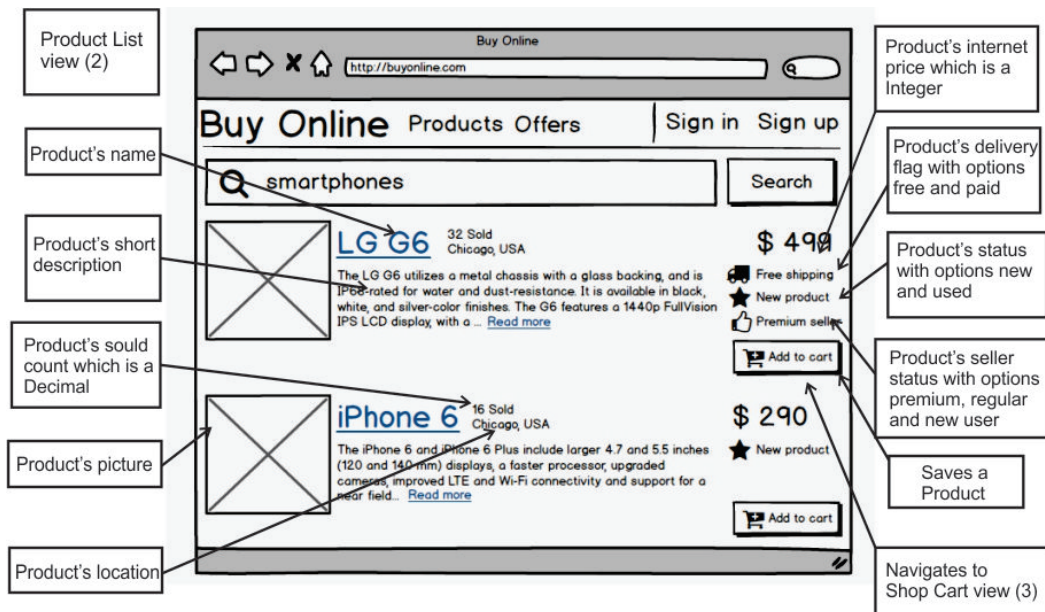


Figura 5.1: Mockup descripto utilizando gramática usuario final

5.1. Implementación de la gramática usuario final

Para lograr una correcta especificación del catálogo, se tuvo que implementar una gramática que determine la manera en que se coordinan las palabras para formar las anotaciones y defina las reglas que la componen. Dicha implementación consistió en escribir la gramática con el objetivo que permita hacer un reconocimiento del lenguaje de las anotaciones que forman parte del catálogo. La misma tuvo que hacerse tanto del lado cliente (Mockup Editor) como del lado del servidor (Mockup Processor). Ver Figura 8.2.

Para definir dicha gramática, se utilizaron 2 librerías que permiten generar analizadores de texto en los lenguajes de programación necesarios para el desarrollo de la herramienta :

- PEGJs para el lenguaje JavaScript, utilizada en el Frontend (ME).
- ANTLR para el lenguaje JAVA, utilizada en el Backend (MP).

Ambas librerías posibilitaron definir las reglas sintácticas que deben cumplirse para que una anotación sea válida y reconocer el lenguaje definido en el catálogo especificado

anteriormente. Las dos tecnologías fueron descritas y presentadas las secciones 4.1.4 y 4.1.5 respectivamente.

A continuación se van a presentar algunas reglas que se utilizaron para crear la gramática de usuario final, donde se va a mostrar la forma en que deben ser escritas. Estas reglas permiten definir sintácticamente la forma en que se escriben las anotaciones presentadas en el catálogo.

5.1.1. Gramática PEGJS

Esta gramática se utiliza en el editor web, la cual permite etiquetar los widgets que forman el mockup y asociar el mismo con la anotación utilizada. Además posibilita inicializar la estructura de datos a partir del diagrama de clases definido en la Figura 4.1 dependiendo del tipo de anotación. Se van a presentar la forma en que se escriben las reglas, con un ejemplo para cada una.

```
1 tag = mockupName/ listClase/ atributoClase/ associationNavigate/  
    subclass/ navigates/ openPopUp / saveClass/ deleteClass/  
    triggers  
2  
3 number= [0-9]  
4 Word = [a-zA-Z]+  
5 tipoNombreMockup = w : (Word _? !"view" Word/Word)  
6  
7 // Ejemplo --> Login view (2)  
8 mockupName= nameMockup:tipoNombreMockup referenciaVista: (" view (  
    "number")"){  
9     return{  
10        tag:{  
11            widget: {  
12                customData: nameMockup.toString().split(",").join(  
                    ""))+ referenciaVista.join("")  
13            },  
14            expresion: 'nombreMockup'  
15        }  
}
```

```
16     }
17 }
18 // Ejemplo --> Navigates to Login view (2)
19 navigates= navigate: ("Navigates to"_) destino: mockupName{
20     return{
21         tag:{
22             widget: {
23                 customData: navigate.toString().split(",").join("")
24                     + destino.tag.widget.customData,
25                 link: true
26             },
27             expression: 'navegacion'
28         }
29     }
30
31 entidad=(alpha*)" 's'
32 optional = w: (Word _? !fraseFinal Word / Word)
33 // Ejemplo --> User's email which is a String
34
35 atributoClase= clase: entidad _ atributo: optional _ fraseFinal: (
36     fraseFinal)? parametro: (paramFinal){
37     return{
38         tag:{
39             widget: {
40                 customData: clase.toString().split(",").join("")+
41                     " +atributo.toString().split(",").join("") +
42                     (fraseFinal? " "+fraseFinal.valor.split(",").
43                     join("") : " which is a String") + (parametro
44                     ? parametro.valor : '')
45             },
46             expression: parametro? parametro.expression : fraseFinal
47                 ? fraseFinal.expression : 'tipoDato'
48         }
49     }
50 }
```



```
44 }  
45 fraseFinal =(fraseTipoDato/fraseValor/attributeRequired/minValue/  
    maxValue/betweenValue/matchesRegular)
```

5.1.2. Gramática ANTLR

Esta gramática se utiliza del lado del servidor, para darle soporte a las reglas que forman cada tipo de inconsistencia a detectar. Dichas reglas van a ser explicadas en la sección 7.3.1. Es semanticamente equivalente a la generada en el punto anterior, dado que esta formada por las mismas anotaciones. Se van a presentar la forma en que se escriben las reglas, con un ejemplo para cada una.

```
1 grammar Mockups;  
2  
3 Letras: [a-zA-Z]+;  
4 Number : '0'..'9'+;  
5 WS : [ \r\t\n]+-> skip ;  
6 palabra: Letras + WS | WS + Letras | Letras;  
7  
8 customData: navegacion | nombreMockup | listaClase |  
    atributoEntidad paramOptional? | propiedadNavegacion |  
    herenciaClase | popUp | atributoRequerido | atributoMinimo |  
    atributoMaximo | expresionRegular | guardarClase |  
    eliminarClase | triggers | EOF;  
9  
10 descripcion: palabra+;  
11 listaClase: ('A ' | 'a ') + (Number| WS) + ('list of'| WS) +  
    descripcion;  
12 navegacion: 'Navigates to' + nombreMockup;  
13 nombreMockup: descripcionMockup + 'view' + '('+Number+ ')';  
14 entidad: Letras + '\s';  
15 atributoEntidad: entidad + descripcion | entidad + descripcion +  
    fraseTipoValor;  
16 fraseTipoValor: 'which is a' + tipoDato | 'with options' + valor+;
```

6. Caracterización de conflictos de requerimientos en Aplicaciones Web

Durante la especificación de requerimientos, puede haber casos en los que dos o más escenarios que reflejen la misma lógica de negocio difieren sutilmente entre sí produciendo una inconsistencia. Cuando estas inconsistencias se basan en comportamientos contradictorios, estamos ante un conflicto de requerimientos [9]. Los conflictos se caracterizan por las diferencias en las características de los objetos, los conflictos lógicos (lo que se espera) o temporales (cuando se espera) entre las acciones, o incluso la diferencia de la terminología que crea ambigüedad. En este análisis se hará hincapié en la navegación de aplicaciones web, así como las peculiaridades de interacción del usuario que no están cubiertos en la caracterización tradicional de los conflictos de requerimientos [9]. En consecuencia, se proporciona una interpretación de cada tipo de conflicto en el ámbito de la aplicación Web, utilizando ejemplos simples pero ilustrativos. En una aplicación pueden aparecer diferentes tipos de conflictos de requerimientos. Por un lado, los conflictos estructurales representan una diferencia en los datos que se espera que sean presentados en una página web por diferentes actores. Un ejemplo de un conflicto estructural puede producirse cuando un interesado solicita que algunos datos se muestren en una página Web que contradiga el requerimiento de otros interesados. Un stakeholder podría esperar una descripción de contenido de producto como una etiqueta de solo lectura, mientras que otro puede esperar que el contenido sea una lista de artículos empaquetados con una descripción general que contradiga el primer requerimiento.

Dos requerimientos de aplicación Web pueden contradecir la forma en que la aplicación se comporta produciendo conflictos de navegación, por ejemplo que tiene

un solo nodo fuente pero objetivos distintos de navegación. Los nodos objetivo son diferentes, pero el evento que desencadena la navegación y las protecciones de condición son los mismos, produciendo una ambigüedad de tal requerimiento. En términos de mockup, para una secuencia de navegación dada (o ruta) compuesta con interfaces de usuario y navegaciones, hay dos alternativas de navegación activadas por el mismo evento. Por ejemplo, la especificación de un requerimiento puede definir que después de hacer clic en el botón “Comprar” en la interfaz de usuario del producto, se presenta un carro de compras. Por otro lado, otro modelo de requerimiento, puede requerir que la misma navegación tenga como objetivo la interfaz de usuario “Método de Pago”, que permite seleccionar un método de pago en lugar de presentar el carro de compras.

Un conflicto semántico ocurre cuando el mismo objeto del mundo real se describe con términos diferentes. Esta situación puede generar un falso negativo en el proceso de detección de conflictos, ya que un conflicto no puede ser detectado y nuevos términos se introducen en el espacio del sistema aumentando así su complejidad. Como consecuencia, el mismo objeto de dominio se modela en dos entidades que tienen terminología diferente.

Por ejemplo, un sitio de comercio electrónico puede definir erróneamente dos entidades que representan el mismo concepto: Bien y Producto. El primero puede definir los atributos nombre, precio, valorización y contenido, mientras que el último está compuesto por identificador, precio, rango y composición. A pesar de ser entidades diferentes y tener diferentes propiedades, se alinean ontológicamente muy bien y tienen el mismo objetivo subyacente: proporcionar una representación para un elemento que se puede comprar a través de la aplicación de comercio electrónico.

7. Detección y resolución de conflictos

El enfoque propuesto comprende los siguientes pasos, representados en la Figura 7.1, los pasos 1 y 2 ya forman parte de cualquier proceso de desarrollo; por lo tanto, la contribución comienza en el paso 3:

1. Recolección de requerimientos: Utilizando técnicas bien conocidas de obtención de requerimientos tales como reuniones, encuestas, Desarrollo de Aplicaciones Conjuntas (JAD), etc, se produce una Especificación de Requerimientos de Software (normalmente en lenguaje natural). En el caso de un proceso de desarrollo ágil subyacente, una descripción más breve se suele producir con historias de usuario [5]; casos de uso se utilizan a menudo en un estilo de proceso unificado.
2. Modelado de requerimientos: Usar mockups para representar los requerimientos de una manera que pueda ser mostrada a las partes interesadas para su futura validación.

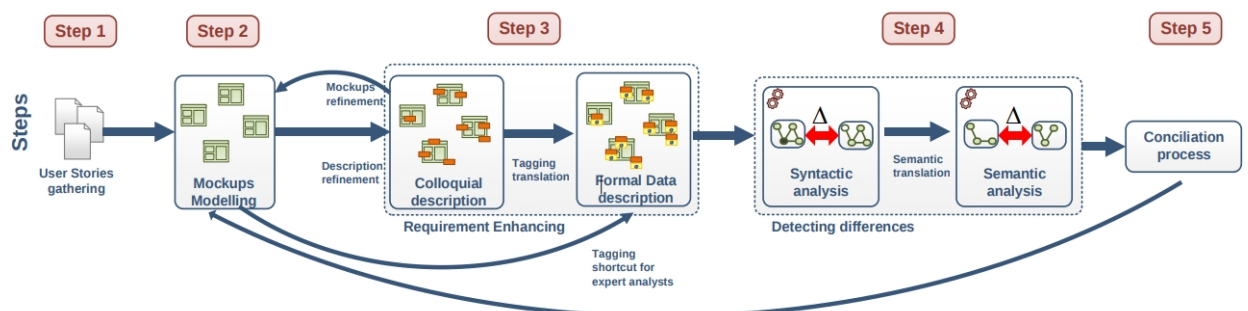


Figura 7.1: El proceso general para detectar conflictos de requerimientos

3. Mejora de requerimientos: Mejora de mockups producidas con una descripción o etiquetas de tal manera que cualquier pieza de datos está bien descrita y sin ambigüedades. A menudo, los mockups dibujan una interfaz de usuario dada con ejemplos de valores de datos, pero no señalan la fuente, el objeto que posee los datos presentados, las restricciones de formato, las reglas de validación, entre otras cosas. Este paso puede lograrse realizando una descripción de los datos coloquiales. Es decir, utilizando características de comentarios de las herramientas de mockups seleccionadas, los analistas adjuntan una descripción a una información usando términos coloquiales siguiendo las pautas sugeridas en este trabajo. Posteriormente, las descripciones se procesan utilizando una solución de procesamiento de lenguaje natural que tiene como resultado etiquetas para la información documentada.
4. Análisis: este es un pasodoble que considera mockups etiquetados para detectar inconsistencias o conflictos. Se realiza la comparación de 2 versiones de un mismo mockup, las cuales pueden pertenecer a historias de usuarios diferentes. Por lo tanto, la diferencia de tiempo entre dichas historias, pueden hacer que los requerimientos nuevos, o variaciones de los existentes generen conflictos o inconsistencias sobre el proyecto. Los modelos SUI con sus anotaciones se comparan en dos niveles diferentes: sintáctico y semántico
 - a) Análisis estructural del modelo de requerimientos Web: mediante una comparación algebraica de modelos, se detectan conflictos estructurales y de navegación candidatos. Además, se evalúan las rutas de navegación para comprobar su coherencia.
 - b) Análisis semántico: se analizan los conflictos de candidatos y se detectan equivalencias semánticas. Para cada conflicto candidato, tanto los requerimientos nuevos como los comprometidos se traducen a una forma mínima, utilizando un constructor atómico para detectar diferencias semánticas. Se

detectan equivalencias semánticas entre requerimientos para advertir a los analistas de requerimientos de posibles conflictos.

5. Proceso de conciliación: una vez confirmada la existencia de un conflicto, debemos comenzar a conciliar los requerimientos. Este proceso exige el establecimiento de un canal de comunicación entre las partes interesadas en el conflicto. Articulación Desarrollo de aplicaciones [12] y la valorización de los requerimientos [2] son herramientas que permiten hacer frente a la tarea de conciliación.
6. Refinamiento: Cuando se confirma un conflicto, debe realizarse algún ajuste y afinación para eliminar el conflicto detectado y alcanzar un estado consistente.

El proceso se aplica iterativamente cada vez que surge un nuevo conjunto de requerimientos. El nuevo conjunto entrante se comprueba con cada uno de los ya consolidados del espacio del sistema.

En las subsecciones siguientes se presenta un ejemplo en ejecución donde se describe cada uno de estos pasos.

7.1. Modelado y simulaciones de historiales de usuario (pasos 1 y 2)

Con el fin de describir de manera clara y precisa el proceso mencionado, se va a utilizar como ejemplo corriente el desarrollo y la extensión de un sitio de comercio electrónico. En las siguientes secciones, los ejemplos se centrarán en dos conjuntos de requerimientos: los recogidos en la primera versión de la aplicación, y los que surgen en la siguiente iteración durante la evolución de la aplicación. El primer conjunto incluye los requerimientos de “Registrar usuario” y “Listado de productos”. La segunda está compuesta por: “Detalle de producto”, “Carrito de Compras”.

Después de que los analistas de software comprendan las necesidades iniciales de los clientes, pueden comenzar a dibujar los mockups para describir de manera

informal como se explorará la aplicación y se utilizarán posteriormente en un proceso de refinación. Los mockups pueden ser modelados utilizando cualquier herramienta en la que el analista tiene experiencia (por ejemplo, Balsamiq¹ o Pencil²). En la Figura 1.1 se muestra un ejemplo de mockups que se pueden construir con tales herramientas.

7.2. Mejoramiento de los requerimientos (paso 3)

Los mockups se utilizan como una herramienta para validar la interpretación de los requerimientos con las partes interesadas. Describen situaciones que usan la apariencia de la interfaz de usuario con datos ilustrativos que ejemplifican escenarios de la vida real. Cuando se utilizan mockups, los analistas se aprovechan del hecho de que el lenguaje que utilizan, los widgets de interfaz de usuario, están libres de jerga (a diferencia de los artefactos de requerimientos textuales) y representan un lenguaje común entre los actores y las partes interesadas [20], [16]. Sin embargo, en algunos casos los requerimientos expresados en los mockups no son suficientemente formales, por ejemplo, en la Figura 1.1, la etiqueta de precio \$ 499 no define claramente como se calcula este precio y su naturaleza (desde el punto de vista comercial). Y los analistas pueden entender este valor como un precio regular o precio de Internet. Una vez que se realiza el modelado de los mockups, los mismos deben digitalizarse para poder ser procesados. Para lograrlo en este trabajo, cada mockup debe ser expresado en formato HTML, donde posteriormente va a poder ser importado desde la herramienta para poder realizar el proceso de detección de conflictos. Con el fin de evitar reducir el nivel de ambigüedad, el enfoque propuesto ofrece realizar una descripción de datos coloquial utilizando la gramática de usuario final presentada en la sección 5.

¹ Balsamiq Mockups - <https://balsamiq.com/products/mockups/>

² Pencil Project - <http://pencil.evolus.vn/>

7.2.1. Descripción de datos coloquiales

Los mockups a menudo utilizan un escenario de la vida real definido con datos ilustrativos para describir cual será la experiencia de usuario de la aplicación (UX). Con el fin de mejorar la descripción de los elementos para resolver la falta de formalidad en el mockup, en este paso usamos la Gramática de Usuario Final (Sección 5) que se centra en describir la fuente de información, el formato y las relaciones de información. Durante este paso, los analistas deben enriquecer los elementos de los mockups con una descripción coloquial basada en plantillas. Para este propósito, los mockups se formalizan con una instancia subyacente del modelo SUI (donde cada widget se asigna a un objeto de clase Widget) y la anotación está representada por una instancia de Tag (Ver Figura 4.1).

Su principal ventaja es una descripción que puede ser fácilmente entendida por los usuarios finales y, en consecuencia, permite una mejor validación de los requerimientos. Cada expresión de descripción debe coincidir con una plantilla específica con marcadores de posición bien definidos facilitará más tarde automatizar el procesamiento.

En la Figura 7.2, podemos ver como se documentó un mockup que representa el requerimiento “Detalle de producto”. Los analistas pueden encontrar otros elementos que también deberían describirse. En la figura, las etiquetas ubicadas a la derecha representan las etiquetas almacenadas como metadatos de comentario, que son proporcionadas por la mayoría de las herramientas de mockup digital. Se utilizara esta representación visual para describir metadatos durante el resto del documento. Una de las etiquetas de la Figura 7.2 (etiquetado) es la expresión “Product ’s category with options home, beauty, music and electronics”. A partir de dicha expresión, podemos extraer la entidad de negocio Producto que tiene un atributo denominado categoría que tiene cuatro valores posibles: hogar, belleza, música y electrónica.

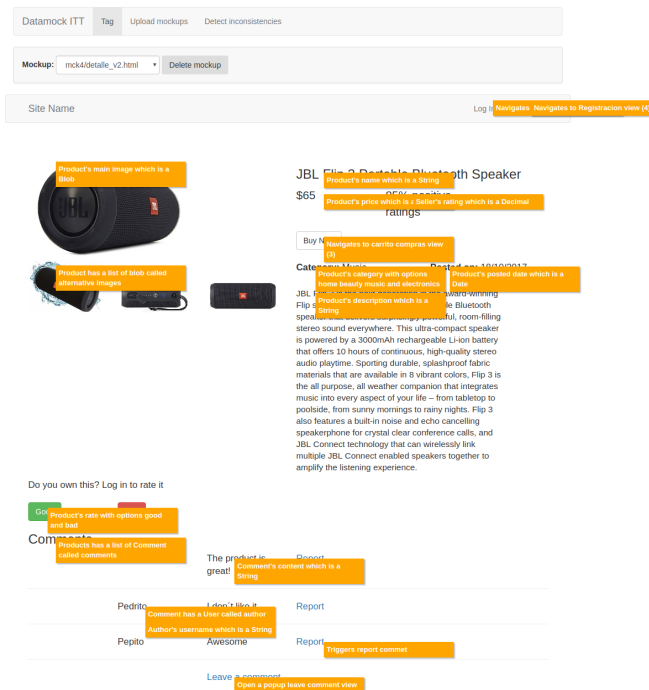


Figura 7.2: Mockup descrito con gramática de usuario final utilizando la herramienta

7.3. Detección de diferencias (paso 4) y conciliación (paso 5)

A continuación, se introducirán un conjunto de heurísticas que ayudan a resolver conflictos estructurales y de navegación y que se han implementado como un catálogo de refactorización en el soporte de la herramienta.

- **Análisis Sintáctico:** Un conflicto candidato surge cuando el conjunto de diferencias sintácticas entre los modelos de requerimientos no está vacío. Estas diferencias pueden ser consecuencia en el uso de dos widgets para describir la misma información y, finalmente, una diferencia de configuración en un elemento como los valores de propiedades de un widget. Esta situación puede surgir cuando dos partes interesadas diferentes tienen diferentes puntos de vista de una sola funcionalidad, o cuando la evolución de un requerimiento contradice uno original. Al anotar mockups con las etiquetas de SUI podemos razonar sobre la especifica-

ción. La detección de conflictos estructurales puede implementarse mediante una operación de comparación entre mockups, con el fin de detectar diferencias de construcciones de elementos.

A la hora de analizar los widgets hay que considerar que pueden existir diferencias entre dos mockups, pero que dichas diferencias no sean conceptualmente dispares, si no que sean falsos positivos. Puede ser el caso de los widgets de tipo button o enlaces, ambos expresan un mismo fin pero se escriben de diferente forma. Para evitar esto, antes de realizar el análisis de conflictos y que se ejecuten las reglas que los validan, se transforman todos los widget button a enlaces a través de una regla Drools. La cual tiene mayor prioridad que las demás y modifica los objetos para todos los widgets que cumplan dicha condición. A continuación se muestra su definición:

```
1 rule "Reemplazo button x link"
2   salience 10
3   //conditions
4   when
5     $tag: Tag (type=="Button" && (this.expression=="
6         navegacion" || this.expression=="popUp"))
7   then
8     //actions
9     $tag.setType("A");
10    update($tag);
11 end
```

La regla evalúa todas las anotaciones de un mockup que están asociadas a un widget de tipo botón, que representen una navegación o activación de PopUp y las reemplaza por un link con el objetivo de descartar posibles inconsistencias o conflictos candidatos que no son tales. Específicamente en la descripción del código de la regla, en la línea 2 se setea la prioridad de la regla dentro del motor,

teniendo esta mayor que las demás dado que se tiene que ejecutar antes de las que realizan la detección de conflictos. En la línea 5 se controla todos los objetos de tipo Tag que su type es igual a “Button” o su expresion sea navegación o PopUp. Para todos los objetos que cumplan esa condición se actualiza su nuevo type, lo cual se realiza en la línea 9. Por último en la línea 10 se hace una actualización del objeto dentro del contexto del motor de reglas para ratificar el cambio realizado.

- **Análisis semántico:** Durante el proceso para detectar conflictos y advertencias se analizan los mockups para descubrir posibles inconsistencias. Para lograr esto, se define un motor de reglas en Drools donde cada una evalúa un tipo de conflicto diferente. Las mismas se ejecutan sobre los datos para determinar si en las anotaciones asociadas a los widgets existen diferencias, haciendo uso del reconocimiento del lenguaje del catálogo a partir de la gramática definida en la sección 5. Los conflictos deben ser resueltos con las soluciones que se proponen, en cambio las advertencias se pueden ignorar para que no sea considerada un conflicto. Estas resoluciones forman parte de un proceso de conciliación entre las partes interesadas, donde se exige un canal de comunicación entre los involucrados en el conflicto. La detección de los conflictos como de advertencias, requieren la intervención del analista para la toma de decisiones, ya sea para resolver el conflicto o ignorar la advertencia.

7.3.1. Tipos de conflictos a detectar

A continuación se pasa a describir, enumerar y ejemplificar las reglas para las cuales la herramienta considera que existe conflicto entre widgets que pertenecen a diferente mockups. Cada regla detecta un tipo de conflicto o inconsistencia, se va a explicar las condiciones que deben cumplirse y el código Drools que se ejecuta para determinar si existe conflicto y/o inconsistencia.

- **Diferente tipo de dato para el mismo atributo entidad:** Cuando se encuentran dos anotaciones que hacen referencia al mismo par atributo entidad, pero que las

The screenshot shows the Datamock ITT interface. At the top, there are tabs for 'Datamock ITT', 'Tag', 'Upload mockups', and 'Detect inconsistencies'. Below this, there are two mockup selection fields: 'Mockup 1: mck3/detalles_v1.html' and 'Mockup 2: (select a mockup)'. There are 'Validate' and 'Reset' buttons. Below the mockup selection, there is a 'Logo' field, a search bar, and a 'Login' button.

The main content area displays a product listing for 'Paul McCartney' with the following details:

- Format:** Audio CD
- Price:** \$65.55 (with a warning: 'Product's price which is a Float')
- Category:** Music
- Product Details:**
 - Audio CD:** (November 4, 2014)
 - Number of disc:** 2
 - ASIN:** B00M20ZLQK
 - Location:** Buenos Aires
 - Posted On:** December 12

There are 'Buy Now' and 'Report' buttons on the right side of the product listing.

Figura 7.3: Mockup A conflicto diferente tipo de dato

The screenshot shows the Datamock ITT interface. At the top, there are tabs for 'Datamock ITT', 'Tag', 'Upload mockups', and 'Detect inconsistencies'. Below this, there are two mockup selection fields: 'Mockup 1: mck4/detalle_v2.html' and 'Mockup 2: (select a mockup)'. There are 'Validate' and 'Reset' buttons. Below the mockup selection, there is a 'Site Name' field, a search bar, and 'Log In' and 'Sign Up' buttons.

The main content area displays a product listing for 'JBL Flip 3 Portable Bluetooth Speaker' with the following details:

- Price:** \$65 (with a warning: 'Product's price which is an Integer ratings')
- Category:** Music
- Posted on:** 18/10/2017

There is a 'Buy Now' button on the right side of the product listing.

Figura 7.4: Mockup B conflicto diferente tipo de dato

mismas están definidas con distinto tipo de dato para dicho par. Es decir, que se quiso especificar al mismo atributo entidad, pero en cada mockup se definió diferentes tipo de dato. Supongamos que tenemos 2 mockups etiquetados con la gramática de usuario final, como las que se ven en la Figura 7.3 y Figura 7.4 donde ambas representan el mismo requerimiento.

Si se realiza la validación de inconsistencias y conflictos entre los 2 mockups, obtenemos el resultado que se muestra en la Figura 7.5

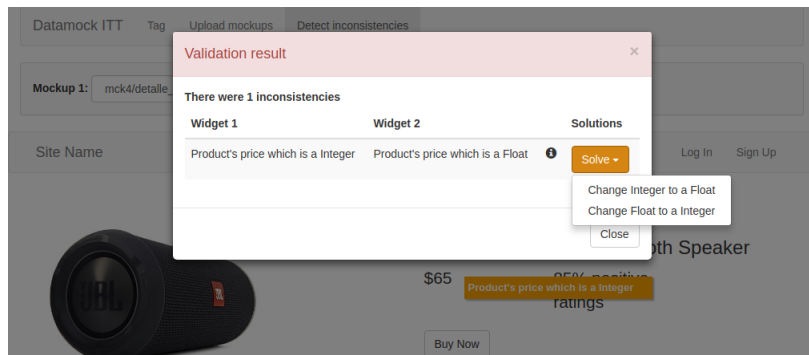


Figura 7.5: Resultado de detección de conflictos entre mockup A y B diferente tipo de dato

Se detecta que existe un conflicto de tipo de dato entre los mockups, dado que ambas especifican el atributo precio del producto, pero con diferente tipo.

La obtención de este resultado, es consecuencia de la ejecución de la regla Drools que en base a la estructura de datos detecta este tipo de conflicto.

A continuación se expone su código:

```

1 rule "Tag de diferente tipo de dato"
2   when
3     //conditions
4     $contr1: Tag (tipoDato!=null, entidad!=null, atributo
5                 !=null)
6     $contr2: Tag (tipoDato!=$contr1.tipoDato && this!=
7                 $contr1 && atributo== $contr1.atributo && entidad
8                 ==$contr1.entidad && $contr1.mockupId!=mockupId &&
9                 atributo!=null && entidad!=null && tipoDato!=null
10                )
11    not InconsistenciaTipoDato (tag1.tagId ==
12                               $contr1.tagId, tag2.tagId==$contr2.tagId)
13    not InconsistenciaTipoDato (tag2.tagId==$contr1.tagId
14                               ,tag1.tagId==$contr2.tagId)
15  then
16    //actions
17    $contr1.setCustomDataDetails($contr1.customData());
18    $contr2.setCustomDataDetails($contr2.customData());

```

```
12         insert(new InconsistenciaTipoDato($contr1,$contr2));  
13     end
```

Lo que realiza la regla es verificar las condiciones que se encuentran en la sentencia when y en caso de que se cumplan realizar las acciones que se especifican después del then. En este caso, crea un objeto de la clase InconsistenciaTipoDato el cual contiene los 2 tags que forman parte del conflicto.

Específicamente en la descripción del código de la regla, se nombran dos variables llamadas contr1 y contr2 que representan dos objetos de la clase Tag. Donde en la línea 4 y 5 se controla que los atributos tipo de dato, entidad y atributo sean diferente entre ambos objetos, y pertenezcan a diferente mockups. En la línea 6 y 7 se valida que ya no exista un objeto conflicto de tipo InconsistenciaTipoDato entre ambos objetos de tipo Tag. En la línea 10 y 11 se setean propiedades pertenecientes a los tags. Por último en la línea 12 se inserta y crea el objeto de tipo InconsistenciaTipoDato dentro del contexto del motor de reglas.

- Botones/Links con igual nombre pero diferente destino: Cuando se encuentran dos anotaciones que describen una navegación, donde el nombre del link son iguales pero cada una activa un destino diferente. Esto representa que en un mockup existe un link con un destino y en un mockup posterior el mismo link tiene un destino diferente al mockup original. Esto requiere la participación del analista para resolver el conflicto.

Supongamos que tenemos 2 mockups etiquetados con la gramática de usuario final, como las que se ven en la Figura 7.6 y Figura 7.7 donde ambas representan el mismo requerimiento.

Si se realiza la validación de inconsistencias y conflictos entre los 2 mockups, obtenemos el resultado de la Figura 7.8

Se detecta que existe un conflicto de navegación entre mockups, dado que el mismo botón tiene diferentes destinos para cada una. Como resultado de este análisis, se le ofrece al analista el refactoring para resolver el conflicto.

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck1/registrar_v1.html Mockup 2: (select a mockup) Validate Reset

Site Name Log In Sign Up

Sign Up

Name

E-mail

Password

Confirm Password

Birthday

Sign Up Navigates to Home view (2)

Already have an account? [Log In](#)

Figura 7.6: Mockup A conflicto de navegación

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck2/registrar_v2.html Mockup 2: (select a mockup) Validate Reset

Sign Up

Name

Last Name

Birthday

Address

Username

E-mail

Password

Confirm Password

Sign Up Navigates to Login view (1)

Figura 7.7: Mockup B conflicto de navegación

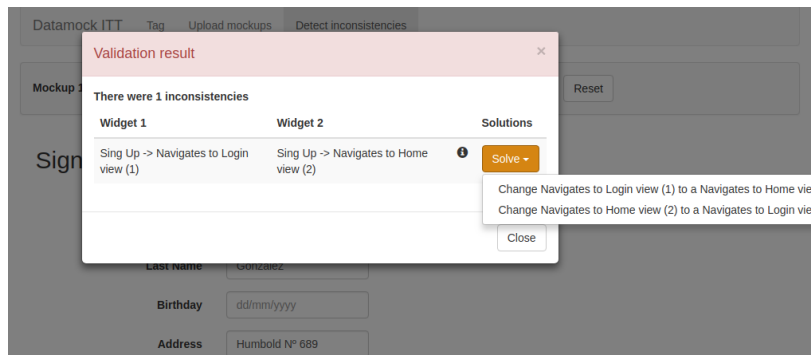


Figura 7.8: Resultado de detección de conflictos entre mockup A y B conflicto de navegación

La obtención de este resultado, es consecuencia de la ejecución de la regla Drools que en base a la estructura de datos detecta este tipo de conflicto. A continuación se expone su código:

```

1 rule "Botones con el mismo nombre y diferente destino"
2   when
3     //conditions
4     $contr1: Tag (type=="A" && destinoBoton!=null && (
5       esLink || esPopUp))
6     $contr2: Tag (text==$contr1.text && destinoBoton!=
7       $contr1.destinoBoton && type=="A" && (esLink ||
8       esPopUp) && $contr1.mockupId!=mockupId && this!=
9       $contr1 && destinoBoton!=null )
10    not InconsistenciaBotonDestino (tag1.tagId ==
11      $contr1.tagId, tag2.tagId==$contr2.tagId)
12    not InconsistenciaBotonDestino (tag2.tagId==
13      $contr1.tagId,tag1.tagId==$contr2.tagId)
14  then
15    //actions
16    $contr1.setCustomDataDetails($contr1.text+" -> "+
17      $contr1.customData());
18    $contr2.setCustomDataDetails($contr2.text+" -> "+
19      $contr2.customData());
20    insert(new InconsistenciaBotonDestino($contr1,$contr2
21      ));

```



```
13  
14 end
```

Lo que realiza la regla es verificar las condiciones que se encuentran en la sentencia when y en caso de que se cumplan realizar las acciones que se especifican después del then. En este caso, crea un objeto de la clase InconsistenciaBotonDestino el cual contiene los 2 tags que forman parte del conflicto.

Específicamente en la descripción del código de la regla, se nombran dos variables llamadas contr1 y contr2 que representan dos objetos de la clase Tag. Donde en la línea 4 y 5 se controla que los tags esten asociados a un widget de tipo link (type=="A") y las propiedades de estos tengan diferente destino y mismo texto, donde ambos tags pertenezcan a diferente mockups. En la línea 6 y 7 se valida que ya no exista un objeto conflicto de tipo InconsistenciaBotonDestino entre ambos objetos de tipo Tag. En la línea 10 y 11 se setean propiedades pertenecientes a los tags. Por último en la línea 12 se inserta y crea el objeto de tipo InconsistenciaBotonDestino dentro del contexto del motor de reglas.

- Mismo atributo pero distinta entidad: Cuando se encuentran dos anotaciones que hacen referencia al mismo atributo, pero que ambas pertenecen a diferentes entidades. Es decir, en ambos mockups existe el mismo atributo, pero se utilizó diferente sujeto. Esto generalmente puede ocurrir, cuando se quiere representar una misma idea, pero se utilizan palabras diferentes como sinónimos o existen distinto puntos de vista sobre una misma propiedad. Esto requiere la participación del analista para resolver el conflicto.

Supongamos que tenemos 2 mockups etiquetados con la gramática de usuario final, como las que se ven en la Figura 7.9 y Figura 7.10 donde ambas representan el mismo requerimiento.

Si se realiza la validación de inconsistencias y conflictos entre los 2 mockups, obtenemos el resultado de la Figura 7.11

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck1/registrar_v1.html Mockup 2: (select a mockup) Validate Reset

Site Name Log In Sign Up

Sign Up

Name Login's username which is a String

E-mail Login's username which is a String

Password Login's password which is a String

Confirm Password Login's passwordConfirmation which is a String

Birthday dd/mm Login's birthday which is a String

Already have an account? [Log In](#)

Figura 7.9: Mockup A conflicto mismo atributo distinta entidad

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck2/registrar_v2.html Mockup 2: (select a mockup) Validate Reset

Sign Up

Name Maria User's username which is a String

Last Name Gonzalez

Birthday dd/mm User's birthday which is a String

Address Humboldt N° 689

Username mgonzalez

E-mail User's email which is a String

Password ***** User's password which is a String

Confirm Password ***** User's passwordConfirmation which is a String

Figura 7.10: Mockup B conflicto mismo atributo distinta entidad

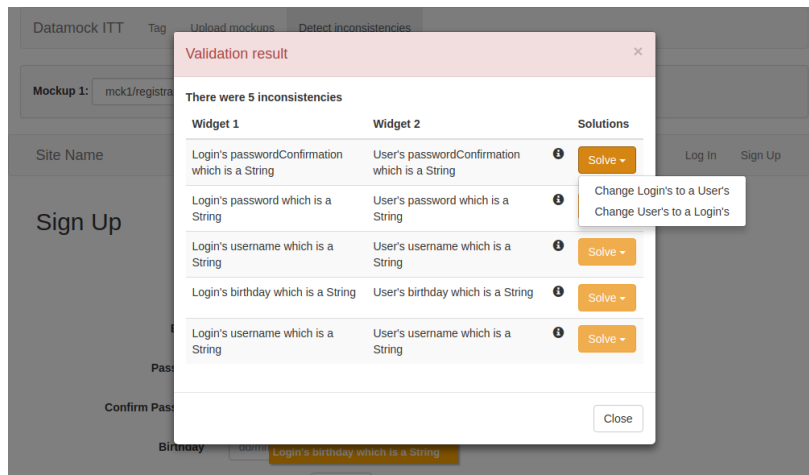


Figura 7.11: Resultado de detección de conflictos entre mockup A y B conflicto mismo atributo distinta entidad

Se detecta que existen conflictos de entidades entre los mockups, dado que para los mismos atributos se especificaron diferentes entidades. Como resultado de este análisis, se le ofrece al analista los refactoring para resolver los conflictos.

La obtención de este resultado, es consecuencia de la ejecución de la regla Drools que en base a la estructura de datos detecta este tipo de conflicto. A continuación se expone su código:

```

1 rule "Mismo atributo, diferente entidad"
2     when
3         //conditions
4         $contr1: Tag (entidad!=null, atributo!=null)
5         $contr2: Tag (atributo==$contr1.atributo, entidad!=
6             $contr1.entidad, this != $contr1, $contr1.mockupId
7             !=mockupId, entidad!=null, atributo!=null)
8         not InconsistenciaEntidad (tag1.tagId ==
9             $contr1.tagId, tag2.tagId==$contr2.tagId)
10        not InconsistenciaEntidad (tag2.tagId==$contr1.tagId,
11            tag1.tagId==$contr2.tagId)
12    then
13        //actions
14        $contr1.setCustomDataDetails($contr1.customData());

```

```
11         $contr2.setCustomDataDetails($contr2.customData());
12         insert(new InconsistenciaEntidad($contr1,$contr2));
13
14     end
```

Lo que realiza la regla es verificar las condiciones que se encuentran en la sentencia when y en caso de que se cumplan realizar las acciones que se especifican después del then. En este caso, crea un objeto de la clase InconsistenciaEntidad el cual contiene los 2 tags que forman parte del conflicto.

Específicamente en la descripción del código de la regla, se nombran dos variables llamadas contr1 y contr2 que representan dos objetos de la clase Tag. Donde en la línea 4 y 5 se controla que el atributo entidad sea diferente pero que el atributo sea el mismo, donde ambos tags pertenezcan a diferente mockups. En la línea 6 y 7 se valida que ya no exista un objeto conflicto de tipo InconsistenciaEntidad entre ambos objetos de tipo Tag. En la línea 10 y 11 se setean propiedades pertenecientes a los tags. Por último en la línea 12 se inserta y crea el objeto de tipo InconsistenciaEntidad dentro del contexto del motor de reglas.

- Mismo valor asignado a widget, contenido en diferente tipo: Cuando se encuentran dos widgets los cuales ambos tienen un mismo valor asignado, pero dichos widgets son de distinto tipo. Comúnmente en los mockups, a determinados widget se les asigna un valor como contenido, para ejemplificar la forma en que tienen que ser utilizados o simplemente para que represente una apariencia más verídica de su uso. Esto representa que se utilizó el mismo valor en mockups diferentes, pero en ambos el tipo de widget que las contiene no son iguales. Esto requiere la participación del analista para resolver el conflicto.

Supongamos que tenemos 2 mockups etiquetados con la gramática de usuario final, como las que se ven en la Figura 7.12 y Figura 7.13 donde ambas representan el mismo requerimiento.

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck1/registrar_v1.html Mockup 2: (select a mockup) Validate Reset

Site Name Log In Sign Up

Sign Up

Name

E-mail mail@ User's email which is a String

Password

Confirm Password

Figura 7.12: Mockup A conflicto mismo valor diferente tipo de widget

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck2/registrar_v2.html Mockup 2: (select a mockup) Validate Reset

Sign Up

Name

Last Name

Birthday

Address

Username

E-mail mail@ User's email which is a String

Password

Figura 7.13: Mockup B conflicto mismo valor diferente tipo de widget

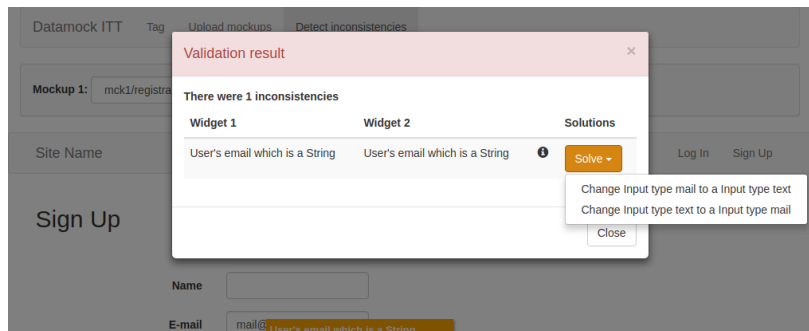


Figura 7.14: Resultado de detección de conflictos entre mockup A y B conflicto mismo valor diferente tipo de widget

Si se realiza la validación de inconsistencias y conflictos entre los 2 mockups, obtenemos el resultado de la Figura 7.14

Se detecta que existe un conflicto, dado que existen widget con el mismo valor (mail@mail.com) asociado a una anotación, pero en ambos mockups se utilizaron diferentes tipo de widget. Específicamente, un widget es de tipo texto de entrada (input text), y otro un campo de texto de tipo mail (input mail). Como resultado de este análisis, se le ofrece al analista el refactoring para resolver el conflicto.

La obtención de este resultado, es consecuencia de la ejecución de la regla Drools que en base a la estructura de datos detecta este tipo de conflicto. A continuación se expone su código:

```

1 rule "Misma texto valor diferente TipoWidget"
2   when
3     //conditions
4     $contr1: Tag (this.valor() != null)
5     $contr2: Tag (this.type != $contr1.type && this.valor()
6                 == $contr1.valor() && this != $contr1 &&
7                 $contr1.mockupId != mockupId && this.valor() != null)
8     not InconsistenciaTipoWidget (tag1.tagId ==
9                                   $contr1.tagId, tag2.tagId == $contr2.tagId)
10    not InconsistenciaTipoWidget (tag2.tagId ==
11                                  $contr1.tagId, tag1.tagId == $contr2.tagId)
12  then

```

```
9      //actions
10     $contr1.setCustomDataDetails($contr1.customData());
11     $contr2.setCustomDataDetails($contr2.customData());
12     insert(new InconsistenciaTipoWidget($contr1,$contr2));
13 end
```

Lo que realiza la regla es verificar las condiciones que se encuentran en la sentencia when y en caso de que se cumplan realizar las acciones que se especifican después del then. En este caso, crea un objeto de la clase InconsistenciaTipoControl el cual contiene los 2 tags que forman parte del conflicto.

Específicamente en la descripción del código de la regla, se nombran dos variables llamadas `contr1` y `contr2` que representan dos objetos de la clase `Tag`. Donde en la línea 4 y 5 se controla que el atributo valor del widget sean iguales y el tipo de widget sean distintos, donde ambos tags pertenezcan a diferente mockups. En la línea 6 y 7 se valida que ya no exista un objeto conflicto de tipo `InconsistenciaTipoControl` entre ambos objetos de tipo `Tag`. En la línea 10 y 11 se setean propiedades pertenecientes a los tags. Por último en la línea 12 se inserta y crea el objeto de tipo `InconsistenciaTipoControl` dentro del contexto del motor de reglas.

- **Misma especificación de atributo entidad con diferentes parámetros:** Cuando se encuentran dos anotaciones que hacen referencia al mismo atributo entidad, al final de la anotación se puede agregar un parámetro opcional. Los posibles parámetros de un atributo, son determinar que el atributo es requerido o especificar una expresión regular que valide el formato que puede tener determinado widget. Cuando este parámetro es diferente para el mismo atributo entidad, existe un conflicto, dado que no se especifica de la misma manera.

Supongamos que tenemos 2 mockups etiquetados con la gramática de usuario final, como las que se ven en la Figura 7.15 y Figura 7.16 donde ambas representan el mismo requerimiento.

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck1/registrar_v1.html Mockup 2: (select a mockup) Validate Reset

Site Name Log In Sign Up

Sign Up

Name

E-mail

Password User's password which is a String. It matches [A-Za-z]

Confirm Password

Figura 7.15: Mockup A conflicto mismo atributo diferente parámetros

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck2/registrar_v2.html Mockup 2: (select a mockup) Validate Reset

Sign Up

Name

Last Name

Birthday

Address

Username

E-mail

Password User's password which is a String. It matches ^[a-zA-Z0-9_-]*\$

Confirm Password

Figura 7.16: Mockup B conflicto mismo atributo diferente parámetros

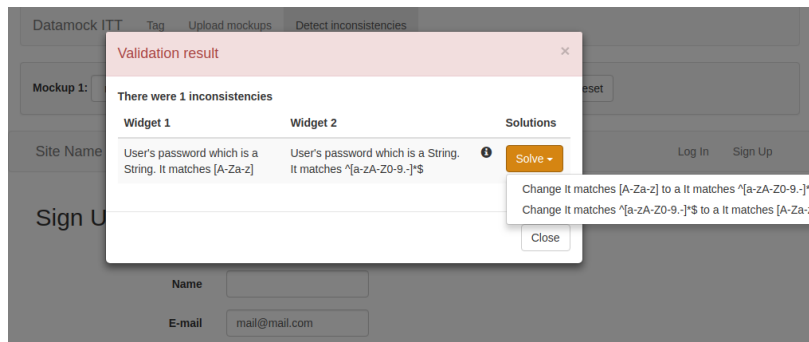


Figura 7.17: Resultado de detección de conflictos entre mockup A y B conflicto mismo atributo diferente parámetros

Si se realiza la validación de inconsistencias y conflictos entre los 2 mockups, obtenemos el resultado de la Figura 7.17

Se detecta que existe un conflicto, dado que existen dos anotaciones para el mismo atributo entidad, pero en ambos mockups dichas anotaciones están acompañadas de parámetros diferentes. Específicamente, difieren en la expresión regular que validan al atributo, donde en un mockup el formato del password acepta solo letras y en otra es acepta letras y números. Como resultado de este análisis, se le ofrece al analista el refactoring para resolver el conflicto.

La obtención de este resultado, es consecuencia de la ejecución de la regla Drools que en base a la estructura de datos detecta este tipo de conflicto. A continuación se expone su código:

```

1 rule "Misma anotacion, diferente parameters"
2   when
3     //conditions
4     $contr1: Tag (atributoClase!=null, parameters!=null)
5     $contr2: Tag (atributoClase==$contr1.atributoClase,
6                 this != $contr1, $contr1.mockupId!=mockupId,
7                 $contr1.parameters!=parameters, atributoClase!=
8                 null, parameters!=null)
9     not InconsistenciaParameters (tag1.tagId ==
10    $contr1.tagId, tag2.tagId==$contr2.tagId)

```

```
7         not InconsistenciaParameters (tag2.tagId==
           $contr1.tagId,tag1.tagId==$contr2.tagId)
8     then
9         //actions
10        $contr1.setCustomDataDetails($contr1.customData());
11        $contr2.setCustomDataDetails($contr2.customData());
12        insert(new InconsistenciaParameters($contr1,$contr2))
           ;
13 end
```

Lo que realiza la regla es verificar las condiciones que se encuentran en la sentencia when y en caso de que se cumplan realizar las acciones que se especifican después del then. En este caso, crea un objeto de la clase InconsistenciaParameters el cual contiene los 2 tags que forman parte del conflicto.

Específicamente en la descripción del código de la regla, se nombran dos variables llamadas contr1 y contr2 que representan dos objetos de la clase Tag. Donde en la línea 4 y 5 se controla la propiedad atributoClase sean iguales pero que se haya especificado diferentes parámetros, donde ambos tags pertenezcan a diferente mockups. En la línea 6 y 7 se valida que ya no exista un objeto conflicto de tipo InconsistenciaParameters entre ambos objetos de tipo Tag. En la línea 10 y 11 se setean propiedades pertenecientes a los tags. Por último en la línea 12 se inserta y crea el objeto de tipo InconsistenciaParameters dentro del contexto del motor de reglas.

7.3.2. Tipos de advertencias a detectar

A continuación se pasa a describir, enumerar y ejemplificar las reglas para las cuales la herramienta considera que existe un warning entre diferentes mockups. Esto no necesariamente tiene que ser un conflicto de requerimientos, por lo que el analista puede ignorar dichas advertencias.

The screenshot shows the Datamock ITT interface. At the top, there are navigation tabs: "Datamock ITT", "Tag", "Upload mockups", and "Detect inconsistencies". Below the tabs, there are two dropdown menus for "Mockup 1" (selected as "mck1/registrar_v1.html") and "Mockup 2" (selected as "(select a mockup)"). There are "Validate" and "Reset" buttons. Below this, there is a "Site Name" field and "Log In" and "Sign Up" links. The "Sign Up" section contains four input fields: "Name", "E-mail" (with "mail@mail.com" entered), "Password", and "Confirm Password". A warning message "User's password which is a String" is displayed in an orange box over the "Password" field.

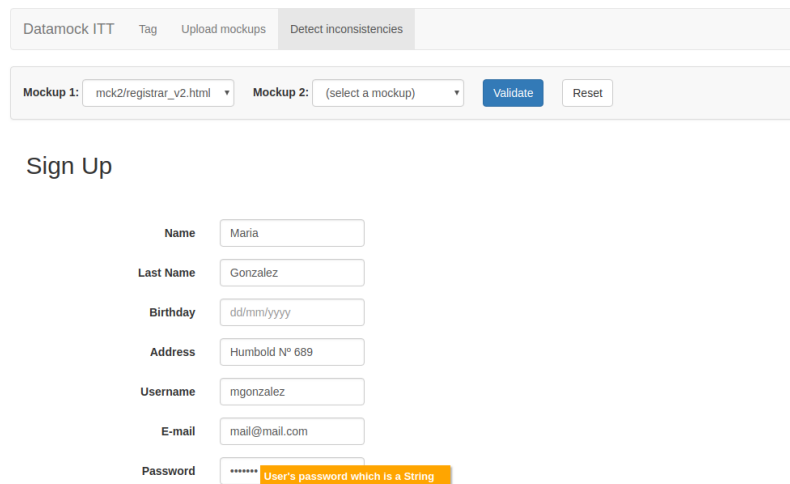
Figura 7.18: Mockup A warning mismo anotación diferente tipo de widget

- Misma anotación relacionados a diferentes tipo de widget: Cuando se encuentran dos widgets, en el cual los dos están asociados a una misma anotación, pero el tipo de widget utilizado no es el mismo en los correspondientes mockups. Esto significa que se quiso describir a un elemento de una misma forma (utilizando la misma anotación), pero se representó a los widgets de diferente forma (utilizando otro tipo). Esto es observado con una advertencia, no necesariamente tiene que ser una equivocación en la especificación de requerimientos. En caso de serlo, se le ofrece al analista posibles soluciones, o simplemente puede ignorar las recomendaciones presentadas.

Supongamos que tenemos 2 mockups etiquetados con la gramática de usuario final, como las que se ven en la Figura 7.18 y Figura 7.19 donde ambas representan el mismo requerimiento.

Si se realiza la validación de inconsistencias y conflictos entre los 2 mockups, obtenemos el resultado de la Figura 7.20

Se detecta una advertencia, dado que para la misma anotación en mockups distintos está asociada a widget de diferente tipo. Específicamente un widget es de tipo texto de entrada (input text) y otro de tipo texto password (input password), es decir los caracteres que se escriban van a estar ocultos.



The screenshot shows the 'Detect inconsistencies' tab in the Datamock ITT interface. At the top, there are navigation tabs: 'Datamock ITT', 'Tag', 'Upload mockups', and 'Detect inconsistencies'. Below these, there are two dropdown menus for 'Mockup 1' (set to 'mck2/registrar_v2.html') and 'Mockup 2' (set to '(select a mockup)'). There are 'Validate' and 'Reset' buttons. The main content is a 'Sign Up' form with the following fields:

- Name: Maria
- Last Name: Gonzalez
- Birthday: dd/mm/yyyy
- Address: Humboldt N° 689
- Username: mgonzalez
- E-mail: mail@mail.com
- Password: ***** (with a warning message: "User's password which is a String")

Figura 7.19: Mockup B warning mismo anotación diferente tipo de widget

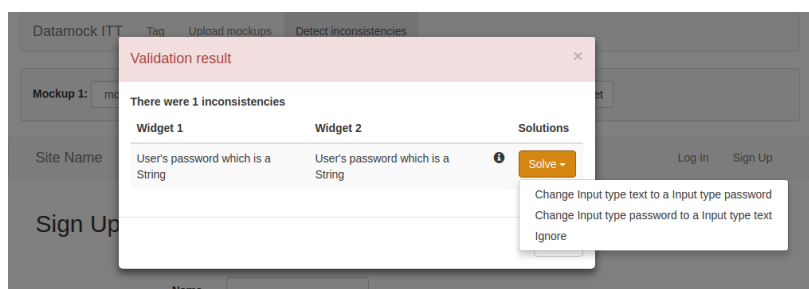


Figura 7.20: Resultado de detección de conflictos entre mockup A y B warning mismo anotación diferente tipo de widget

La obtención de este resultado, es consecuencia de la ejecución de la regla Drools que en base a la estructura de datos detecta este tipo de conflicto. A continuación se expone su código:

```
1 rule "Misma anotacion diferente TipoWidget"
2   when
3     //conditions
4     $contr1: Tag (customData!=null, customData not memberOf
5       this.ignores)
6     $contr2: Tag (this.type != $contr1.type, customData==
7       $contr1.customData, this != $contr1,
8       $contr1.mockupId!=mockupId, customData!=null &&
9       customData not memberOf this.ignores)
10    not WarningTipoWidget (tag1.tagId == $contr1.tagId,
11      tag2.tagId==$contr2.tagId)
12    not WarningTipoWidget (tag2.tagId == $contr1.tagId,
13      tag1.tagId==$contr2.tagId)
14    not InconsistenciaTipoWidget (tag1.tagId ==
15      $contr1.tagId, tag2.tagId==$contr2.tagId)
16    not InconsistenciaTipoWidget (tag2.tagId ==
17      $contr1.tagId, tag1.tagId==$contr2.tagId)
18  then
19    //actions
20    $contr1.setCustomDataDetails($contr1.customData());
21    $contr2.setCustomDataDetails($contr2.customData());
22    insert(new WarningTipoWidget($contr1,$contr2));
23  end
```

Lo que realiza la regla es verificar las condiciones que se encuentran en la sentencia when y en caso de que se cumplan realizar las acciones que se especifican después del then. En este caso, crea un objeto de la clase WarningTipoWidget el cual contiene los 2 tags que forman parte de la advertencia.

Específicamente en la descripción del código de la regla, se nombran dos variables llamadas contr1 y contr2 que representan dos objetos de la clase Tag. Donde en

la línea 4 y 5 se controla que el atributo de la anotación utilizado del catálogo sean iguales pero el atributo del tipo de widget sea diferente, donde ambos tags pertenezcan a diferente mockups. Además se controla que el warning no haya sido ignorado previamente para no detectarlo innecesariamente. En la línea 6 y 7 se valida que ya no exista un objeto warning de tipo `WarningTipoWidget` entre ambos objetos de tipo `Tag`. En el caso específico de esta regla, en la línea 8 y 9 se debe validar que no exista un conflicto de `InconsistenciaTipoWidget`, dado que ambas reglas validan sobre el tipo de widget asociado a un tag, pero el conflicto tiene una consideración mayor en comparación a una advertencia. En la línea 12 y 13 se setean propiedades pertenecientes a los tags. Por último en la línea 14 se inserta y crea el objeto de tipo `WarningTipoWidget` dentro del contexto del motor de reglas.

- Diferentes botones o links con distinto nombre referencian al mismo destino: Cuando se encuentran dos widgets, donde diferentes botones/links hacen referencia al mismo destino, pero los widgets tienen diferente nombre. Es decir, existen dos referencias a un mismo destino, pero dichas referencias que se encuentran en mockups diferentes, no están nombradas de la misma forma. Esto es observado con una advertencia, no necesariamente tiene que ser una equivocación en la especificación de requerimientos. En caso de serlo, se le ofrece al analista posibles soluciones, o simplemente puede ignorar las recomendaciones presentadas.

Supongamos que tenemos 2 mockups etiquetados con la gramática de usuario final, como las que se ven en la Figura 7.21 y Figura 7.22 donde ambas representan el mismo requerimiento.

Si se realiza la validación de inconsistencias y conflictos entre los 2 mockups, obtenemos el resultado de la Figura 7.23

Se detecta una advertencia, dado que existen dos widgets (botón/link) que tienen el mismo destino, pero que el nombre es diferente en ambos mockups. Específicamente un widget tiene nombre Log In y otro Sign In, y ambos tienen destino de

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck1/registrar_v1.html Mockup 2: (select a mockup) Validate Reset

Site Name Log In Navigates to Login view (2)

Sign Up

Name

E-mail

Password

Confirm Password

Birthday

Figura 7.21: Mockup A warning diferentes botones mismo destino

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck2/registrar_v2.html Mockup 2: (select a mockup) Validate Reset

Sign Up

Name

Last Name

Birthday

Address

Username

E-mail

Password

Confirm Password

Sing Up Navigates to Login view (2)

Figura 7.22: Mockup B warning diferentes botones mismo destino

Datamock ITT Tag Upload mockups Detect inconsistencies

Mockup 1: mck1/registrar_v1.html

Site Name

Sign Up

Name

Validation result

There were 1 inconsistencies

Widget 1	Widget 2	Solutions
Log In -> Navigates to Login view (2)	Sing Up -> Navigates to Login view (2)	Solve -

- Change Log In to a Sing Up
- Change Sing Up to a Log In
- Ignore

Figura 7.23: Resultado de detección de conflictos entre mockup A y B warning diferentes botones mismo destino

navegación al mockup de Login.

La obtención de este resultado, es consecuencia de la ejecución de la regla Drools que en base a la estructura de datos detecta este tipo de conflicto. A continuación se expone su código:

```
1 rule "Botones con el mismo destino y diferente nombre"
2   when
3     //conditions
4     $contr1: Tag ( texto()!=null, this.type=="A", (esLink
5       || esPopUp), customData not memberOf this.ignores)
6     $contr2: Tag (text!=$contr1.text && destinoBoton==
7       $contr1.destinoBoton && texto()!=null && this.type==
8       "A" && esLink==$contr1.esLink && esPopUp==
9       $contr1.esPopUp && $contr1.mockupId!=mockupId &&
10      this!=$contr1 && customData not memberOf
11      this.ignores)
12    not WarningDestinoBoton (tag1.tagId == $contr1.tagId,
13      tag2.tagId==$contr2.tagId)
14    not WarningDestinoBoton (tag2.tagId==$contr1.tagId,
15      tag1.tagId==$contr2.tagId)
16  then
17    //actions
18    $contr1.setCustomDataDetails($contr1.text+" -> "+
19      $contr1.customData());
20    $contr2.setCustomDataDetails($contr2.text+" -> "+
21      $contr2.customData());
22    insert(new WarningDestinoBoton($contr1,$contr2));
23  end
```

Lo que realiza la regla es verificar las condiciones que se encuentran en la sentencia when y en caso de que se cumplan realizar las acciones que se especifican después del then. En este caso, crea un objeto de la clase WarningDestinoBoton el cual contiene los 2 tags que forman parte de la advertencia.

Específicamente en la descripción del código de la regla, se nombran dos variables llamadas `contr1` y `contr2` que representan dos objetos de la clase `Tag`. Donde en la línea 4 y 5 se controla que los tags estén asociados a un widget de tipo link donde los atributos `texto` sean iguales pero el destino del mismo sean distintos, donde ambos tags pertenezcan a diferente mockups. Además se controla que el warning no haya sido ignorado previamente para no detectarlo innecesariamente. En la línea 6 y 7 se valida que ya no exista un objeto warning de tipo `WarningDestinoBoton` entre ambos objetos de tipo `Tag`. En la línea 10 y 11 se setean propiedades pertenecientes a los tags. Por último en la línea 12 se inserta y crea el objeto de tipo `WarningDestinoBoton` dentro del contexto del motor de reglas.

A su vez se controla que el warning no haya sido ignorado previamente para no detectarlo innecesariamente.

7.4. Evolución de requerimientos de gestión

Durante diferentes ciclos de recopilación de requerimientos, el conjunto de requerimientos evoluciona donde, eventualmente, existen algunos que describen un solo flujo de trabajo desde diferentes puntos de vista. Por ejemplo, en una historia de usuario, el analista puede omitir describir completamente un mockup dado por razones de simplicidad, pero en otra historia de usuario se acordó en que dicho mockup debe mostrar cierta información y debe satisfacer cierta navegación específica. Por lo tanto, apoyamos esta situación al generar incrementalmente una vista consolidada de mockups con su navegación y anotaciones, siempre y cuando lleguen nuevos requerimientos. Además, después de consolidar los requerimientos, cada mockup definido en un caso de uso es una proyección de interfaz consolidada de la aplicación.

En la Figura 7.24, se muestra una secuencia de nuevos requerimientos y como se fusionan construyendo un modelo consolidado del sistema. Por motivos de simplicidad presentamos un esquema simplificado. En el lado derecho, se muestra como un modelo de sistema completo se construye de forma incremental teniendo en cuenta las mejoras

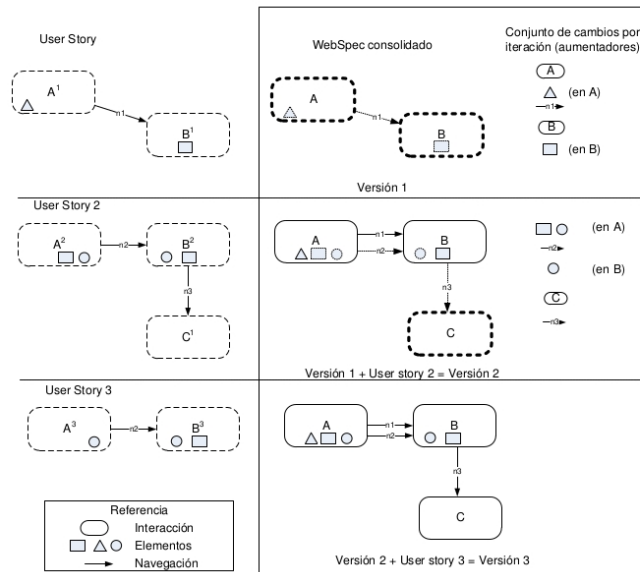


Figura 7.24: Impacto de los requerimientos en el modelo de la aplicación

proporcionadas por los últimos requerimientos. El modelo agregado combina su versión anterior con la nueva contribución. En la versión 1, el modelo consolidado se construye a partir de la historia de usuario 1 porque es un modelo vacío.

En la segunda iteración, el modelo comprende todo el Mockup C, los widgets definidos por el Mockup A2 y una navegación de B a C llamada n2. Y, finalmente, en la tercera iteración, el modo agregado no introduce ningún cambio ya que cada elemento definido en la historia de usuario 3 ya fue aportado por la historia de usuario 1 y 2.

A medida que se van sucediendo las iteraciones, se validan requerimientos entre historias de usuario haciendo uso de la herramienta, por lo que al final de la iteración se cuenta con una versión consolidada de los requerimientos de ambas historias. Dicha versión pasa a ser un posible mockup definitivo para el sistema a desarrollar, hasta la siguiente historia de usuario en la cual pueden surgir nuevos requerimientos o sufrir cambios los existentes.

8. Arquitectura de la herramienta

Se decidió desarrollar una aplicación web para aprovechar las ventajas de sus características, entre las que destacan acceder desde cualquier lugar que se disponga Internet usando un navegador web; haciendo uso de tecnologías como HTML5 y JavaScript, permitiendo que la herramienta sea compatible con múltiples dispositivos y plataformas que se utilizan hoy en día. Mencionadas cualidades, son importantes debido a que posibilita el acceso a la herramienta a un número grande de usuarios.

Las aplicaciones web se basan en el paradigma de cliente-servidor y como su nombre lo indica se compone de dos partes. Por un lado, el usuario, quien ejecuta una aplicación en su equipo local, denominado programa cliente. En el caso específico de las aplicaciones web, el cliente es el navegador quien se encarga de ponerse en contacto con un servidor remoto (generalmente mediante Internet) para solicitar el servicio deseado. Por otro lado, el servidor responderá a las solicitudes mediante un programa que se está ejecutando. Este último se denomina programa servidor, el cual está a la espera de solicitudes por parte del cliente. En la herramienta, del lado del servidor se ejecuta, un programa desarrollado en el lenguaje de programación JAVA, específicamente, utilizando un framework web llamado Spring. Para el lado del cliente, se aprovecharon las últimas tecnologías en el desarrollo web como HTML5 y JavaScript, que facilita la utilización de librerías que simplifican la programación, como jQuery y PEGJs. Dichas tecnologías fueron presentadas y descritas en la sección 4.1

En la Figura 8.1 se muestra un diagrama de las tecnologías, clasificandolas por el lado donde se ejecuta (cliente o servidor) y el lenguaje de programación que utiliza (JAVA o JavaScript).

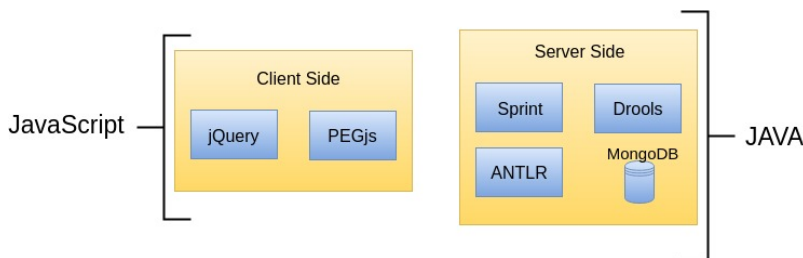


Figura 8.1: Representación de las tecnologías utilizadas

Desde el punto de vista funcional, la herramienta está compuesta por dos partes principales: Mockup Editor (ME) y Mockup Processor (MP). El Mockup Editor (ME) es un editor del lado del cliente implementado en JavaScript que permite importar los mockups diseñados previamente para poder ser utilizados para realizar la detección de conflictos o inconsistencias. Además, esta parte de la herramienta permite también colocar las anotaciones previamente introducidas sobre los widgets definidos en el mockup. Siendo más específico, el ME es de hecho una herramienta de etiquetado que almacena las anotaciones como atributos de datos incrustados en el código HTML del mockup. Todos los cambios realizados en el ME se sincronizan en el lado del servidor mediante una API RESTful.

El Mockup Processor (MP) es la parte de la herramienta que detecta los conflictos de requerimientos y propone refactorings cuando es posible. Cuando se invoca desde la vista, realiza las siguientes tareas:

- Posee una API RESTful que permite la comunicación con el editor, tanto para crear mockups, eliminarlos, agregar/eliminar anotaciones y realizar la detección de conflictos.
- Crea un modelo SUI en la memoria de los widgets y anotaciones almacenados en el mockup.
- Detecta conflictos y/o inconsistencias sobre los mockups digitalizados, aplicando las reglas Drools especificadas en la sección 7.3

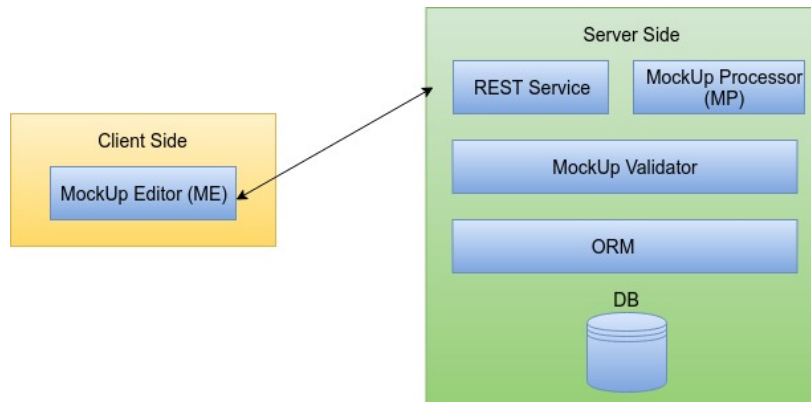


Figura 8.2: Arquitectura de la herramienta desarrollada

En la Figura 8.2 se muestra como esta compuesta la arquitectura de la herramienta, donde se encuentran los componentes descritos y la forma de comunicación entre ellos.

9. Evaluación de la herramienta

En esta sección, se realizará una evaluación del enfoque para medir cuanto ayuda a la comprensión de los mockups siguiendo a Wohlin et al. directrices [29]. En primer lugar, se definen los objetivos, las hipótesis y las variables del experimento. A continuación, se procede a definir métricas y materiales considerados. Después de eso, se detallan temas, instrumentación y métodos de recopilación de datos utilizados. Luego se realizará un análisis de resultados y su implicación. Por último, se consideran las amenazas a la validez de la evaluación.

9.1. Objetivos, hipótesis y variables

En esta investigación, el objetivo es evaluar como los mockups mejorados con las anotaciones del usuario final permiten detectar inconsistencias y/o ambigüedades en la elicitación de requerimientos.

La principal pregunta de investigación (RQ) es: ¿Las anotaciones del usuario final facilitan la detección de errores en mockups? Con el fin de responder a esta pregunta, se diseñó un experimento donde los sujetos fueron invitados a identificar errores de especificación entre mockups que se presentan como contradicciones o ambigüedades; a partir de ahora, Variable de Experimento (EV). Los participantes fueron divididos en dos grupos para las dos versiones del experimento: mockups sin ningún tipo de anotaciones y mockups (Control) con el uso de las anotaciones del usuario final (Tratamiento). Para este RQ, consideramos como hipótesis nula H_0 que la técnica utilizada no presentaba una mejora en rendimiento frente a la otra técnica. Como hipótesis alternativa H_a , se considera que hay una mejora en la media de la exactitud

de las respuestas de los participantes que utilizaron anotaciones del usuario final (μEU) contra mockups sin anotaciones (μMOCK): $\mu\text{MOCK} \leq \mu\text{EU}$.

En este experimento, se enfocó en medir la cantidad de contradicciones encontradas entre diferentes versiones de mockups correspondiente a un mismo requerimiento de software. No se consideró la comparación de rendimiento frente con otro enfoque.

9.2. Métricas y materiales

La tarea de recopilación de requerimientos mediante mockups requiere documentar mockups y comunicarlos, en primer lugar, a las partes interesadas para validar su definición y posteriormente a los desarrolladores para iniciar su desarrollo. Para esta tarea, se ha modelado casos para un sitio de comercio electrónico usando mockups - las cuales eran similares a las mostradas en la Figura 1.1. Las principales funcionalidades consideradas en los casos de uso fueron el registro en el sistema y la visión de los detalles de un producto. Ambas versiones usaron los mismos mockups, pero una de ellas incluía las anotaciones definidas en la sección 5 para mejorar su descripción.

Tanto los resultados de las tareas de modelado como de marcado fueron validados por analistas con vasta experiencia previamente al experimento. Para evaluar la comprensión de los participantes sobre el requerimiento, se le pidió que llenaran un formulario donde debían registrar cada uno de los datos presentes en los mockups, su tipo de datos esperado, cualquier tipo de validación y el widget asociado.

Dado que las mockups representan escenarios utilizando ejemplos en lugar de variables abstractas o marcadores de posición, carecen de cualquier tipo de formalismo por lo que el tipo de datos, las validaciones y cualquier otra especificación es el resultado de la interpretación del lector del mockup. Tanto los mockups como el formulario están disponibles en ¹. Para poder calcular la eficiencia de los participantes, se tuvo que definir cuales eran las respuestas correctas para cada pregunta que se hizo en el cuestionario, de modo que se utilizaron como respuestas de referencia o testigo a la hora de analizar

¹https://www.dropbox.com/sh/nnj8t3fqsx0vtmw/AADWy_PfOnjYPttJOoiaMdrMa?dl=0

los resultados. La elección de las dichas respuestas estuvo a cargo de analistas con amplia experiencia en elicitación de requerimientos donde analizaron los mockups presentados y determinaron las opciones acertadas a las preguntas realizadas a los participantes.

9.3. Asignaturas, Instrumentación y Recolección de Datos

Durante el experimento, los participantes recibieron un formulario y un conjunto de mockups. Los participantes eran 26 desarrolladores de diferentes compañías de software. En promedio, tenían 30 años, 5 años de experiencia en programación y alrededor de 3 años en tareas de análisis de requerimientos. Un grupo de 14 participantes realizó el experimento con mockups anotados con la gramática del usuario final mientras tanto un grupo de 12 participantes realizó el experimento basado en mockups simples. Ellos estaban motivados y comprometidos con el experimento, ya que fueron patrocinados por los CEOs y gerentes que notificaron a los participantes sobre el compromiso de la compañía con el experimento de investigación.

El protocolo del experimento se ejecutó de la misma manera con todos los participantes de ambos grupos y comprendió un flujo de trabajo bien definido. En primer lugar, recibieron una breve introducción al material que debía utilizarse durante el experimento. A continuación, se pidió a los participantes que completaran una encuesta de experiencia, leyeran la descripción del experimento, estudiaran los mockups y completaran el cuestionario.

Para lograr la tarea de procesar los resultados obtenidos, primero se procesan las respuestas y su posterior digitalización. Luego, se utilizan diferentes scripts para calcular precisión [14], recall (exhaustividad), accuracy (exactitud) y una medida que combina la precisión y recall de forma armónica la cual se denomina F1.

9.4. Análisis

Para el análisis de muestras, se aprovechan los conceptos de precisión y recall [14] del campo de investigación de la recuperación de información y se adaptan al experimento para medir la precisión de las respuestas.

Para lograr analizar los resultados de los participantes, se digitalizaron todas las muestras y se volcaron todas las respuestas en una base de datos. En la misma estaban todas las preguntas que fueron realizadas en el cuestionario, y las respuestas de referencia que se utilizaran para determinar la precisión de cada participante. Tener esta información digitalizada permitió calcular elementos Verdadero Positivo (TP), Falso Positivo (FP), Verdadero Negativo (TN) y Falso Negativo (FN) de cada participante y a partir de estas, medir las métricas a evaluar.

Para el análisis fue necesario definir inicialmente los componentes necesarios en las fórmulas de Precision, Recall, F1 y Accuracy:

- Verdadero Positivo (TP): Aquellos elementos que un participante informó correctamente al verificar si su respuesta está incluida en el conjunto de respuestas de referencia válidas. Un ejemplo de este concepto podría considerarse que existe una alarma que detecta fuego, si la alarma suena y existe fuego, la predicción es verdadera y el fuego positivo.
- Falso Positivo (FP): Aquellos elementos que un participante informó incorrectamente, al verificar que su respuesta no estaba incluida en el conjunto de respuestas de referencia válidas. Siguiendo el ejemplo de la alarma, si la alarma suena y no existe fuego, la predicción es falsa dado que se predijo que el fuego era positivo.
- Verdadero Negativo (TN): Aquellos elementos que un participante rechazó correctamente de su conjunto de respuestas. Es decir, las posibles opciones incorrectas que no contesto. En el ejemplo de la alarma, si no hubo fuego y la alarma no sonó, la predicción es verdadera dado que el fuego es negativo.

- Falso Negativo (FN): Aquellos elementos que un participante no informó, pero si estaban dentro del conjunto de respuestas de referencia válidas. Es decir, las respuestas correctas que le faltaron contestar al participante. En el ejemplo de la alarma, si la alarma no sonó, pero si hubo fuego, la predicción es falsa dado que se predijo fuego negativo.

A continuación se presenta la forma en que se calculan las métricas que se van a medir a partir de la información obtenida de las respuestas.

$$\text{Precisión} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$F1 = 2 \cdot \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Por último, todas las muestras se procesaron mediante Mann-Whitney U test [23], que es una técnica de prueba de hipótesis estadística no paramétrica, para evaluar si las medidas repetidas difieren. El análisis se realizó principalmente de manera automatizada utilizando scripts basados en Python que resolvieron el cálculo de la precision, recall, accuracy, F1 y la prueba de hipótesis.

9.5. Evaluación de Resultados e Implicación

Para responder a la pregunta de investigación, se ha evaluado las muestras de los participantes y se determinó el resultado de las mismas en base a las respuestas consideradas correctas para cada pregunta realizada durante el experimento. Se considero un solo criterio a la hora de evaluar las muestras, donde en base a lo contestado se

Evaluation	Metric	Mockup 1				Mockup 2			
		Avg. Mockup	Avg. Annotation	P-value	<0.05	Avg. Mockup	Avg. Annotation	P-value	<0.05
Variable	Precision	0,6108	0,5871	0,365		0,2083	0,3764	0,978	
	Recall	0,1133	0,4279	0,000	√	0,0500	0,3714	0,000	√
	F1	0,1875	0,4464	0,000	√	0,0792	0,3600	0,001	√
	Accuracy	0,5833	0,8657	0,000	√	0,3608	0,8221	0,000	√

Figura 9.1: Resultados de la muestra

computaba si la respuesta estaba o no en el conjunto de respuestas válidas para calcular la métrica de cada participante.

Con los resultados individuales se calculo un promedio general de cada métrica, clasificado de acuerdo si los mockups tenían o no anotaciones y a que mockup pertenecían dichos resultados. En la Figura 9.1, se muestran los resultados obtenidos sobre las métricas y podemos darnos cuenta de que las muestras de los participantes que utilizaron mockups anotados tienen mayor rendimiento en comparación con los participantes que utilizaron mockups sin anotaciones. Siendo el promedio de todas las métricas superior en todos los casos excepto en la precisión. Esto último se debe a que en base a la fórmula de precisión, existieron muchos casos de participantes que utilizaron mockups sin anotaciones que respondieron pocas preguntas, las cuales estaban correctas. Teniendo un buen rendimiento en dicha métrica, pero obviando preguntas que no fueron contestadas, justificando que no tenían información del dominio del mockup para responder dichas preguntas. En cuanto a las demás métricas, el p-value fue menor que el nivel alfa 0,05. Por lo tanto, hay suficiente evidencia para apoyar la hipótesis alternativa.

Adicionalmente, se calcula el tiempo requerido para que cada individuo complete el experimento. Es de destacar que los participantes que extrajeron información de las mockups anotados realizaron mejor desempeño (requirieron menos tiempo) que los que trabajaron sin anotaciones. Para las mockups anotados 1 y 2, le llevó a los participantes en promedio 129 segundos y 105 segundos respectivamente. A la inversa, para las mockups no anotados, requirió 180 segundos y 238 segundos.

La métrica de recall señala que existe una comprensión efectiva de los datos comprometidos por el mockup. Los participantes que utilizaron anotaciones respondieron un mayor número de elementos relevantes que los sujetos que trabajaron con mockups simples. Esto significa que los participantes eran más precisos con la descripción del elemento de dominio, dado que tenían más información para poder detectar contradicciones o ambigüedades. Esta es otra importante indicación de que las respuestas no combinan información válida e inválida reduciendo el ruido en la comunicación entre clientes y analistas. Por ejemplo, sin las anotaciones, los sujetos definieron diferentes entidades comerciales irrelevantes tales como Inicio de sesión, Usuario y Persona para componer los atributos de la interfaz de usuario.

9.6. Amenazas a la validez

Existen varias amenazas a la validez que se consideraron durante el diseño del experimento, las cuales van a ser detalladas a continuación.

Validez de construcción: El experimento fue diseñado para medir como los mockups mejorados con las anotaciones del usuario final permiten detectar inconsistencias y/o ambigüedades durante la elicitación de requerimientos. Con el fin de reducir la complejidad del experimento y la posibilidad de introducción de sesgo, solo se define el método (mockups simples o mockups anotados) como la única variable. El lector debe notar que el enfoque no está siendo comparado con otro enfoque, y, de hecho, está bajo evaluación como la extensión de anotaciones mejora los mockups básicos.

Validez interna: Para evitar cualquier malentendido durante el experimento, se presentó cada material en una breve introducción antes de que los participantes realicen el experimento y durante el experimento, cualquier consulta relacionada con las oraciones fue contestada sin introducir un sesgo en la muestra. Los temas fueron seleccionados al azar y todos ellos estaban trabajando en empresas de software en Argentina. El material proporcionado fue el mismo para todos los participantes. También se comprobó que todos los usuarios tenían conocimientos básicos en aplicaciones de

comercio electrónico (solo usuarios simples) y no han participado en el desarrollo o análisis de requerimientos en ninguna aplicación de este tipo.

Amenazas a la validez externa: Los participantes eran ingenieros de software que han desempeñado el papel de desarrolladores y/o analistas durante su carrera. Aunque su experiencia es diferente, se exponen a las responsabilidades regulares de cualquier profesional de software: reunirse con los clientes, entender los requerimientos, desarrollar el software y cumplir con los plazos de entrega de software. Un experimento más amplio considerando diferentes temas de diferentes culturas que han trabajado en diferentes dominios de negocios mejorará la generalidad de las afirmaciones.

10. Conclusiones y Trabajos Futuros

Los resultados preliminares mostrados en la sección anterior, apoyan la hipótesis alternativa de que el uso de anotaciones sobre los mockups aportan mayor conocimiento de los requerimientos del cliente, elevando el nivel de especificaciones del dominio con el objetivo de reducir los errores y eliminar ambigüedades durante la etapa de desarrollo y elicitación de requerimientos. Si bien estos resultados son favorables, sería útil realizar un ensayo donde participen mayor cantidad de analistas, para aumentar la cantidad de muestras para analizar con mayor profundidad el rendimiento del uso de la herramienta. A su vez sería adecuado que se evalúe el uso de la herramienta dentro de la industria de software, en donde se obtendrían resultados más tangibles de su empleo en proyectos y se conseguiría la valoración por parte de analistas.

Sin embargo, de los resultados analizados previamente, se desprende que se tiene evidencia para rechazar la hipótesis nula planteada, donde se consideraba que no existía una mejora con el uso de anotaciones sobre mockups que permitan detectar inconsistencias y/o ambigüedades en la especificación de requerimientos.

Como posibles trabajos futuros sería interesante de realizar en la herramienta desarrollada, la extensión de el catálogo de etiquetas para el usuario final disponible en los mockups. Dicha extensión podría variar dependiendo en el ámbito donde se aplican, es decir va a depender de las características de los requerimientos del usuario. Esto posibilitaría aun más la expresión sobre los mockups, facilitando a los analistas y desarrolladores tener un conocimiento más cercano del dominio de las necesidades del usuario, pero orientado a un ambiente específico. Permitiendo que el catálogo no sea

restringido y limitado a las anotaciones predefinidas; a su vez las anotaciones pueden ser agrupadas o clasificadas dependiendo de el entorno en donde se utilicen.

Estas subclasificaciones podrían ser por ejemplo orientadas a sistema contable, comercio electrónico, sistema de salud, sistema administrativo, etc. Mencionada extensión de las anotaciones requiere un análisis más profundo del dominio en que se enfocan, dado que va a depender de la información que se quiera conseguir de los mockups y el grado de expresión que se quiera obtener. Esto requeriría la modificación o creación de una nueva gramática que defina las reglas que deben cumplirse para formar las anotaciones que surjan.

A su vez, se podría estudiar ampliar los tipos de conflictos o advertencia a detectar por la herramienta, a partir de las anotaciones disponibles (ya sea existentes o nuevas). La detección de nuevas inconsistencias, demanda la creación de nuevas reglas para cada nuevo conflicto, donde se determinen las condiciones que deben cumplirse para que se produzca. Además se debe estudiar y analizar las posibles formas de resolverlo, para brindarle al analista las opciones a aplicar una vez detectado.

Bibliografía

- [1] K. Altmanninger. Models in conflict - Towards a semantically enhanced version control system for models. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* volume 5002 LNCS, pages 293–304, 2008.
- [2] B. Boehm, P. Grunbacher, and R. Briggs. Developing Groupware for Requirements Negotiations: Lessons Learned. *IEEE Software*, 18(3):46–55, 2001.
- [3] I. S. Brito, F. Vieira, A. Moreira, and R. A. Ribeiro. Handling conflicts in aspectual requirements compositions, 2007.
- [4] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
- [5] M. Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 2009.
- [6] M. J. Escalona and G. Aragón. NDT. A model-driven approach for web requirements. *IEEE Transactions on Software Engineering*, 34(3):377–394, 2008.
- [7] M. J. Escalona and N. Koch. Requirements engineering for web applications: a comparative study. *Journal of Web Engineering*, 2(3):193–212, feb 2003.
- [8] M. J. Escalona, M. Urbietta, G. Rossi, J. A. Garcia-Garcia, and E. R. Luna. Detecting Web requirements conflicts and inconsistencies under a model-based perspective. *Journal of Systems and Software*, 86:3024–3038, 2013.

-
- [9] IEEE. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998*, pages 1–40, 1998.
- [10] G. Kotonya and I. Sommerville. Requirements engineering with viewpoints, 1996.
- [11] D. Leffingwell. Calculating the return on investment from more effective requirements management. *American Programmer*, 10:13–16, 1997.
- [12] A. D. Lucia and A. Qusef. Requirements Engineering in Agile Software Development, 2010.
- [13] E. R. Luna and I. Garrigós. Capture and Evolution of Web Requirements Using WebSpec 2 WebSpec : A DSL to Capture Interactive Web Requirements. *10th International Conference in Web Engineering*, 6189:173–188, 2010.
- [14] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [15] A. Martínez, O. Pastor, J. Mylopoulos, and P. Giorgini. From early to late requirements: A goal-based approach. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4898 LNAI, pages 123–142, 2008.
- [16] K. S. Mukasa and H. Kaindl. An Integration of Requirements and User Interface Specifications. In *6th IEEE International Requirements Engineering Conference*, pages 327–328, Barcelona, sep 2008. IEEE Computer Society.
- [17] M. G. D. Paula, B. Santana, S. Diniz, and J. Barbosa. Using an Interaction Model as a Resource for Communication in Design. *Interface*, pages 1713–1716, 2005.
- [18] R. Ramdoyal and A. Cleve. From Pattern-based User Interfaces to Conceptual Schemas and Back. In *Proceedings of the 30th International Conference on Conceptual Modeling - ER 2011*, pages 247–260, Brussels, 2011.

-
- [19] A. Ravid and D. M. Berry. A Method for Extracting and Stating Software Requirements that a User Interface Prototype Contains. *Requirements Engineering*, 5(4):225–241, 2000.
- [20] J. M. Rivero, J. Grigera, G. Rossi, E. Robles Luna, F. Montero, and M. Gaedke. Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering. *Information and Software Technology*, 56(6):670–687, jan 2014.
- [21] J. M. Rivero, E. Robles Luna, J. Grigera, and G. Rossi. Improving user involvement through a model-driven requirements approach. In *2013 International Workshop on Model-Driven Requirements Engineering (MoDRE)*,, pages 20–29, Rio de Janeiro, 2013.
- [22] D. Rosenberg, M. Stephens, and Collins-Cope. *Agile Development with ICONIX Process—People, Process, and Pragmatism*. 2005.
- [23] S. M. Ross. *Introduction to probability and statistics for engineers and scientists*. Academic Press, Cambridge,MA, USA, 2004.
- [24] K. Schneider. Generating fast feedback in requirements elicitation. In *Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality*, pages 160–174, jun 2007.
- [25] I. Sommerville. *Software Engineering*. 2010.
- [26] R. V. D. Straeten, T. Mens, J. Simmonds, and V. Jonckers. Using Description Logic to Maintain Consistency between UML Models. In *Management*, volume 2863, pages 326–340, 2003.
- [27] D. Turk, R. France, and B. Rumpe. Assumptions underlying agile software-development processes. *Journal of Database Management*, 16(4):62–87, 2005.
- [28] M. Urbietta, M. J. E. Cuaresma, E. R. Luna, and G. Rossi. Detecting Conflicts and Inconsistencies in Web Application Requirements. In A. Harth and N. Koch,

- editors, *Current Trends in Web Engineering - Workshops, Doctoral Symposium, and Tutorials, Held at {ICWE} 2011, Paphos, Cyprus, June 20-21, 2011. Revised Selected Papers*, volume 7059 of *Lecture Notes in Computer Science*, pages 278–288. Springer, 2011.
- [29] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*, volume 15. Kluwer Academic Publishers Norwell, Netherlands, 2000.
- [30] D. Yang, Q. Wang, M. Li, Y. Yang, K. Ye, and J. Du. A survey on software cost estimation in the chinese software industry. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '08*, page 253, 2008.