

EXTENSIÓN Y AUTOMATIZACIÓN DE PRUEBAS PARA UN SISTEMA LEGADO

Pasantía

JHON ANDERSON CARDENAS DIAZ
0932309
Jhon.a.cardenas@correounivalle.edu.co

Napoleón Moreno Rojas
nmoreno@innova4j.com

Oswaldo Solarte
solarte7@gmail.com

Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Programa Académico de Ingeniería de Sistemas
Cali, Junio 11 de 2014

Dedicado a las personas que estuvieron a mi lado de manera incondicional, mi madre, mi novia, mi padrastro y mis hermanos. Sin ellos tal vez hubiera sido posible, pero seguramente no hubiera sido lo mismo.

Jhon Anderson Cardenas Diaz

AGRADECIMIENTOS

Agradecimientos para mis compañeros de trabajo en Innova4j que ayudaron a forjar el trabajo que aquí se desarrolla gracias a sus ayudas y opiniones siempre críticas e imparciales. Agradezco de manera especial a quien me guio inicialmente en la compañía e invirtió gran parte de su tiempo para aclarar las dudas que me aquejaban, Hermes Jiménez analista en Innova4j.

Tabla de Contenido

RESUMEN 1
ABSTRACT 2
1 INTRODUCCIÓN 3
1.1 PLANTEAMIENTO Y FORMULACIÓN DEL PROBLEMA 3
1.2 JUSTIFICACIÓN DEL PROBLEMA 4
1.2.1 Justificación Económica 4
1.2.2 Justificación Social 4
1.2.3 Justificación Académica 5
1.3 OBJETIVOS 5
1.3.1 Objetivo General 5
1.3.2 Objetivos Específicos 5
2 DESARROLLO DEL TRABAJO 6
2.1 ANTECEDENTES O ESTADO DEL ARTE 6
2.2 DESARROLLO TEÓRICO DE LOS OBJETIVOS 8
2.2.1 Integración de nuevo módulo en cambio tecnológico 8
2.2.1.1 Inconvenientes de la actual arquitectura del sistema 9
2.2.1.2 Marco de trabajo usado para la construcción del nuevo módulo 11
2.2.1.3 Herramientas tecnológicas usadas para la construcción del nuevo módulo 13
2.2.1.4 Arquitectura planteada para el nuevo módulo 15
2.2.1.5 Conexión entre el nuevo módulo y el sistema legado 17
2.2.2 Migración de procedimientos pl/sql a Java 20
2.2.2.1 Criterios para seleccionar la selección de los procedimientos a migrar 20
2.2.2.2 Proceso de migración 20
2.2.3 Automatización de pruebas para el sistema legado 25
2.2.3.1 Situación del sistema legado 25
2.2.3.2 Planteamiento de la propuesta 26
2.2.3.3 Inconveniente en la ejecución de la propuesta y su solución 35
3 CONCLUSIONES 41
4 TRABAJO FUTURO 43

5 REFERENCIAS 44
6 ANEXOS 46
6.1 Prueba de concepto - Manual de instalación 46
6.2 Prueba de concepto 46

Índice de Ilustraciones

Ilustración 1 Arquitectura conceptual del sistema actual 10
Ilustración 2 Transformaciones XSL 11
Ilustración 3 Ejemplo básico de uso de MyBatis 14
Ilustración 4 Arquitectura conceptual planteada para el nuevo módulo 16
Ilustración 5 Ejemplo de filtro paso 1 17
Ilustración 6 Ejemplo de filtro paso 2 18
Ilustración 7 Ejemplo de filtro paso 3 19
Ilustración 8 Ejemplo de filtro paso 4 19
Ilustración 9 Ejemplo de polimorfismo en migración parte 1 21
Ilustración 10 Ejemplo de polimorfismo en migración parte 2 22
Ilustración 11 Ejemplo de SQL Dinámico con MyBatis 23
Ilustración 12 Ejemplo de Manejo de transacciones PLSQL 24
Ilustración 13 Ejemplo de uso de Transacciones en Spring 25
Ilustración 14 Esquema conceptual de pruebas automáticas 26
Ilustración 15 Paso 1 para la creación de una prueba. 27
Ilustración 16 Paso 2 para la creación de una prueba. 28
Ilustración 17 Paso 3 para la creación de una prueba. 28
Ilustración 18 Paso 4 para la creación de una prueba. 29
Ilustración 19 Diagrama de clases de pruebas automáticas 30
Ilustración 20 Configuración para la ejecución de un conjunto de pruebas. 33
Ilustración 21 Configuración de ejecución para el proyecto de pruebas 34
Ilustración 22 Uso de plantillas como entradas para las pruebas 36
Ilustración 23 Uso de las anotaciones de JAXB para transformaciones a de Java a XML 38
Ilustración 24 Xml resultante de la transformación de un producto específico 38
Ilustración 25 Método encargado de fabricar clases de prueba 39

RESUMEN

La propuesta de trabajo de grado consiste en el desarrollo de un nuevo módulo para un sistema web legado [1] llamado Sure2Care, esto mediante una metodología de desarrollo ágil [2] y basado principalmente en la tecnología JAVA [3]. Se tiene como finalidad principal para este nuevo módulo, plantear una arquitectura base y ser implementado bajo tecnologías actuales. Todas las funcionalidades del sistema existente deberán permanecer intactas, y las nuevas funcionalidades del módulo a desarrollar deben poder entenderse con el sistema existente.

Junto con este módulo se llevará acabo el planteamiento y desarrollo de un conjunto de pruebas automáticas para el sistema, las cuales serán realizadas de tal forma que interactúen con la aplicación directamente tal como el usuario final, sin que se entienda con el código del sistema ni la base de datos que lo soporta. Esto porque dado que es un sistema legado, se dificulta mucho desarrollar pruebas al sistema con las tecnologías en las que está desarrollado. Otra razón es porque en el futuro este sistema será migrado a otras arquitecturas y tecnologías, en cuya situación las pruebas desarrolladas ahora seguirán teniendo validez (a menos que las funcionalidades cambien).

Palabras Clave: Sistema legado, metodología de desarrollo ágil, pruebas automáticas, migración.

ABSTRACT

The proposed for this work is the development of a new web module for a legacy system [1] called Sure2Care, through an agile development methodology [2] and based mainly on JAVA [3] technology. The main purpose for this new module is to propose a basic architecture for it and be implemented under current technologies. All the functionality of the existing system should remain intact, and the new features of the module to develop should be able an understanding with the existing system.

Together with this module will take place the planning and development of a set of automated tests to the system, which will be carried out in such a way to interact with the application directly as the end user, without it being the system code or database that supports it. This is because since it is a legacy system, are much more difficult to develop testing system in which technologies are developed. Another reason is because in the future this system will be migrated to other architectures and technologies, in which situation now developed tests remain valid (unless it change functionality).

Key Words: Legacy system, agile development methodology, automated testing, migration.

1 INTRODUCCIÓN

1.1 PLANTEAMIENTO Y FORMULACIÓN DEL PROBLEMA

Se plantea y define el problema desde el punto de vista de la empresa cliente (empresa que contrata los servicios de Innova4j) que requiere la realización de lo mencionado anteriormente.

Es evidente que en el transcurso del tiempo, las tecnologías van siendo obsoletas, más en la actualidad, donde la emergencia de nuevas tecnologías se da con mucha más rapidez. Consecuencia directa de esto es que, los sistemas grandes y vitales de las empresas, van teniendo un ciclo de vida cada vez más corto. La mantenibilidad de estos sistemas es bastante baja y a medida que el tiempo transcurre, se vuelven considerablemente más costosos para la empresa que los usa.

Aunque la tecnología es tal vez uno de los factores más incidentes en las problemáticas de un sistema legado, muchas otras son las razones por las que se debería pensar en el reemplazo de estos sistemas, las principales son: la documentación no es adecuada, poca escalabilidad, la mano de obra es cada vez más escasa (las tecnologías obsoletas son poco conocidas), poseen arquitecturas que no dan abasto a las necesidades actuales y por último, las técnicas de software para optimización de espacio y velocidad de ejecución usadas en el momento de la creación de estos sistemas son distintas a las usadas ahora.

El área de negocio de la empresa cliente de Innova4j requiere un nuevo módulo para el sistema existente. Y a causa de las desventajas que traen que el sistema sea legado, han decidido que el nuevo módulo sea desarrollado bajo tecnologías y arquitecturas actuales, y que también pueda ser unido con el sistema que ya está, de tal manera que sea transparente para el usuario final. Se debe alcanzar un estado donde este nuevo módulo constituya una herramienta y punto de apoyo para la empresa, y no un nuevo dolor de cabeza.

Además de esto, no existen pruebas que prueben las funcionalidades del sistema. Cada vez que se cambia algo en la aplicación las consecuencias son problemáticas, porque surgen errores causados por dichos cambios. Muchas son las razones por las que este sistema es tan vulnerable, pero las principales son que es un sistema legado, y que carece de pruebas sistematizadas. La propuesta de automatización de pruebas que se propondrá en este trabajo, trata de pruebas que se entienden directamente con el sistema, en otras palabras simulan ser el usuario final, no entrarán en contacto con el código fuente de la aplicación ni con las base de datos que la soporta.

Tratar a un sistema legado es un desafío de por sí, pero más específicamente existen ciertos desafíos para la ejecución de este trabajo de grado, se mencionan los más notables:

- La nueva arquitectura a plantear debe permitir al nuevo módulo ser integrado al sistema existente. Esto se refiere al intercambio de datos, la persistencia de información y las interfaces gráficas, lo que es dificultoso debido a que la estructura del sistema es obsoleta.
- Muchas de las operaciones realizadas bajo el sistema se encuentran establecidas en la base de datos como procedimientos “pl/sql”. Dependiendo de lo que decida el cliente, las operaciones comunes a otros módulos serán re-utilizadas, o re-desarrolladas. Si se da el último caso, dichos procedimientos deberán ser migrados a Java, lo cual es engorroso si se tiene en cuenta la escasa (o nula) documentación de dichos procedimientos.

- Dado que las pruebas que se realizarán para el sistema se entenderán con la vista de la aplicación, se debe asegurar que cubran en profundidad y en amplitud sus funcionalidades. Y que la arquitectura de pruebas base, sea útil para la continuación con el desarrollo de todas las demás pruebas (esto porque las pruebas que se plantean en este trabajo no pretenden ser todas las del sistema, sino un pequeño conjunto de pruebas que sirva como base para el desarrollo de las demás).

Planteados estos desafíos, surgen preguntas críticas para que el proceso realizado sea exitoso. ¿Qué tecnologías de la actualidad pueden ser usadas para la implementación de la nueva funcionalidad?, ¿Qué metodología o estrategia usar para lograr que la comunicación con la empresa cliente sea lo más clara y concisa posible?, ¿Cómo adaptar la metodología de desarrollo ágil para la realización de este proceso, teniendo en cuenta que se debe pensar en el futuro de la aplicación generando la documentación necesaria para establecer características, cumpliendo los tiempos estipulados y logrando por completo los objetivos?, ¿Cómo establecer la nueva arquitectura de tal manera que la calidad en todo sentido sea la mayor posible?.

1.2 JUSTIFICACIÓN DEL PROBLEMA

Para la justificación económica también se tendrá en cuenta el punto de vista de la empresa cliente.

1.2.1 Justificación Económica

La empresa cliente requiere que el sistema cubra otras ciertas funcionalidades además de para las que fue originalmente hecho, pero los altos costos de mantenimiento de este sistema legado ha hecho que sea necesario que los nuevos desarrollos sean implementados con tecnologías y arquitecturas actuales; además es un sistema vulnerable a cambios, por lo que las tediosas pruebas realizadas cada vez que sufre algún cambio planean ser automatizadas. Estos dos cambios aportarán un valor significativo al sistema desde el punto de vista económico, ya que lo volverá más mantenible y escalable, lo que directamente influye en la disminución de costos y tiempo.

1.2.2 Justificación Social

El compromiso y en general la responsabilidad que conlleva la realización de este trabajo de grado provee a los realizadores experiencia significativa y prueba su capacidad de compromiso, esto sin duda hace que sus vidas como profesionales, desde el punto de vista ético, dé un paso más, en el camino que queda por recorrer. Este trabajo brinda también las bases suficientes para enfrentar proyectos de mayor envergadura, en cuanto a conocimiento técnico, que es también importante en la vida de un profesional integro.

1.2.3 Justificación Académica

Los retos que plantea la realización de este trabajo, hace necesario la profundización de los conocimientos adquiridos durante el proceso de educación universitaria. Y no solo los aprendido en el transcurso de los estudios, si no que incentiva también la búsqueda y adquisición de conocimientos novedosos, que por ser tan específicos, no son enseñados en la universidad. Cabe mencionar que el conocimiento es abstracto, y poco sirve en la vida profesional si no es aplicado, por lo que otra ventaja de este trabajo de grado, en especial en modalidad pasantía, es la posibilidad de poner en práctica lo que se ha aprendido, aprender más, y ver como se hacen las cosas en el mundo real, en el que el idealismo con el que se plantean muchos de los problemas en la academia pierde validez.

1.3 OBJETIVOS

Se presentan los objetivos de este trabajo de grado.

1.3.1 Objetivo General

Lograr que el sistema resultante tenga implantadas las nuevas funcionalidades de acuerdo a las expectativas del cliente y que el conjunto de pruebas base facilite el desarrollo de otras pruebas más adelante que puedan cubrir en profundidad y en amplitud los procesos llevados a cabo en el sistema.

1.3.2 Objetivos Específicos

Los objetivos específicos serán abordados teniendo como guía los desafíos planteados en el planteamiento y la formulación del problema.

- Lograr establecer y desarrollar una arquitectura para el nuevo módulo que además de ser adecuada para las necesidades funcionales de este módulo, permita la conexión con el sistema sin problema alguno.
- Migrar las operaciones necesarias a Java para la lógica del nuevo módulo del sistema.
- Plantear y desarrollar un conjunto de pruebas automatizadas que sirvan como base para que futuras pruebas puedan ser desarrolladas. Esto incluyendo la elección de arquitectura y tecnologías.
- Desarrollar en el transcurso del período de pasantía una monografía y pruebas de concepto que ilustren el trabajo que se realizó en el proceso de pasantía.

Dado que el trabajo desarrollado en la empresa Innova4j es confidencial, el presente documento es una monografía y es, junto a la prueba de concepto, el **resultado esperado** del cuarto objetivo específico. Contiene el aspecto teórico aplicado para realizar cada uno de los primeros tres objetivos.

2 DESARROLLO DEL TRABAJO

2.1 ANTECEDENTES O ESTADO DEL ARTE

A continuación se presentan brevemente algunos trabajos o publicaciones relacionados con la temática principal de este trabajo de grado.

Temas

La primera publicación a mencionar titula *Evaluación de riesgos de seguridad en sistemas legados* [4]. Esta publicación interviene en el proceso en que se estudia el nivel de afectación de un sistema legado sobre la entidad que lo usa, y qué medidas se deben tomar para esto. Resumidamente, resalta la importancia de tener en cuenta que la seguridad de una entidad puede verse amenazada por el uso de un sistema legado, ya sea que su uso sea en mayor o en menor medida. Esto debido a que no importa mucho que un gran sistema esté protegido si uno de sus módulos, por pequeño que sea, es vulnerable. Se pensaba que solo se debía considerar reemplazar o actualizar un sistema legado que fuera grande o influyente para la empresa, pero teniendo lo anterior claro, se puede decir que un sistema legado puede potencialmente afectar a la entidad que lo usa por pequeño o insignificante que este sea.

Se plantea en esta publicación una manera para evaluar el riesgo en seguridad que los sistemas legados pueden causar a una empresa. Como primera medida, se debe listar el conjunto de sistemas heredados, posterior a esto, identificar cuáles de estos sistemas son potencialmente de alto riesgo, luego, para cada sistema que se identificó como potencialmente de alto riesgo, realizar un análisis más profundo de cómo se puede, por medio de este, dañar a la entidad y por último, realizar un plan de acción para mitigar los riesgos que estos proyectos presentan para la empresa. Algo interesante es que, en general, las opciones que pueden ser propuestas en el plan de acción pueden ser: hacer nada (si el sistema no se considera finalmente como un riesgo potencial), modificar el sistema (eliminando vulnerabilidades sin cambiar funcionalidades), mejorar el sistema (agregando o quitando funcionalidades con el fin de mejorar su seguridad) o reemplazando el sistema legado.

En el artículo *Migrating Legacy Systems to Service-Oriented Architectures* [5], se plantea una metodología para la migración de sistemas legados hacia sistemas con arquitectura orientada a servicios. Plantea el papel que desempeña la arquitectura SOA (Service-Oriented Architectures) en la reingeniería que se da en el proceso de la migración de un sistema legado es muy importante, potencializando las mejoras del nuevo sistema que reemplazara al sistema legado. Las características básicas de la arquitectura SOA que hacen de esta un factor importante para esta migración son, según este artículo:

- La separación entre los negocios y la lógica de presentación.
- La relación débilmente acoplada entre los servicios.
- El carácter genérico de los servicios.

Otra publicación importante es *Métodos de testing sobre la ingeniería de requisitos web* [6], la cual muestra maneras para la creación de casos de prueba a partir de los requerimientos del sistema. Plantea que a partir de estos requerimientos se pueden los diferentes caminos que un sistema puede tomar, tanto en transcursos normales, como en transcursos inesperados. La consideración de cada posible camino puede ser convertida en un caso de prueba, facilitando de esta manera dos aspectos importantes, la detección de errores en el sistema mediante el uso de pruebas y con estas guiar el desarrollo del sistema mismo.

Finalmente se referencia una publicación que está fuertemente ligada al desarrollo que se llevó a cabo con la automatización de pruebas para el sistema en cuestión y para su nuevo módulo. La publicación *Automating Functional Tests Using Selenium* [7] describe, de manera general, el uso de la herramienta *Selenium* para la automatización de pruebas funcionales en aplicaciones web y resalta la evolución de *Selenium* con respecto a otras herramientas con fines similares como *Canoo WebTest* o *HTMLUnit* en cuanto a aspectos como el mantenimiento de las pruebas y la posibilidad de escribirlas antes de desarrollar su solución en el sistema. La publicación además presenta una especie de metodología para la creación de las pruebas:

- Primero se codifican las pruebas.
- Luego se realiza el respectivo desarrollo en el sistema para que las pruebas pasen
- Y finalmente se re-factorizan las pruebas

Según la publicación, este proceso iterativo permite definir o mejorar el diseño del sistema y permite la reutilización y no duplicación de las pruebas creadas.

Relación con el trabajo actual

La primera publicación se relaciona con este trabajo de grado en el sentido en que la razón por la que el nuevo módulo esta sobre tecnologías actuales, es porque se espera que el sistema completo sea migrado, y esto se debe a que dicho sistema legado puede afectar seriamente la integridad y confidencialidad de los datos de la compañía que hace uso de él. Son varios los sistemas que intervienen en los procesos de negocio de la compañía, uno de ellos es el sistema que se estudia en este trabajo de grado, y dado que se determinó que dicho sistema es un riesgo potencial, se concluye que las nuevas modificaciones sean sobre nuevas tecnologías y pensadas para una futura migración completa.

La segunda publicación no se relaciona del todo con el trabajo de grado ya que no se estableció una arquitectura orientada a servicios para el nuevo módulo. No se logró realizar una separación o definición clara de un conjunto de servicios específicos que fueran autosuficientes y que pudieran ser consumidos por el nuevo módulo, ya que los procesos de negocio que se llevan a cabo en dicho módulo están muy acoplados entre sí y acoplados con los procesos de negocio que ya existen en el sistema actual. El uso de una arquitectura orientada a servicios para el nuevo módulo hubiera implicado la separación y migración de una cantidad considerable de procesos existentes en el sistema, y era una consecuencia que la empresa dueña del producto no aceptó por razones de coste y tiempo.

La tercera publicación permitió definir un proceso de creación de pruebas efectivo, ya que se analizaron cada uno de los requisitos de manera separada y se crearon pruebas de aceptación para

los mismos, pudiendo ser capaces de mostrar al cliente los avances respaldados en pruebas de aceptación en cada etapa de desarrollo del nuevo módulo. Dado que también se debían crear pruebas para el sistema existente, se tomó un enfoque parecido al que presenta la tercera publicación, solo que en lugar de tomar como entrada los requisitos del sistema original (los cuales no están documentados) se tomaron como entrada los errores reportados a lo largo del soporte que se ha brindado a dicho sistema. Esto permitió la creación de un plan de pruebas que incluían tanto el nuevo módulo, como el sistema ya existente.

Sin duda la cuarta y última publicación referenciada fue la que más sirvió como fuente de referencia conceptual para el desarrollo de las pruebas automáticas con el uso de Selenium, ya que si bien no se usaron todas las demás herramientas que describe la publicación, ni tampoco al pie de la letra su proceso, si se tuvieron en cuenta las afirmaciones que en ella se encontraban, por ejemplo, a mantener cada una de las pruebas autosuficientes, es decir, evitar dependencia entre ellas; también con base a esta publicación surgió la idea de crear un banco de variables agrupándolas dependiendo de la pantalla del sistema a la que pertenecían, esto evitó la duplicación de variables y facilitó que si en algún momento llega a cambiar el valor de alguna variable, por ejemplo, el identificador de un campo de texto, el cambio tuviera efecto en todas las pruebas que usen dicho campo de texto.

2.2 DESARROLLO TEÓRICO DE LOS OBJETIVOS

Se desarrolla a continuación el aspecto teórico de cada uno de los tres primeros objetivos. Se tendrá como punto de partida el sistema legado que se ha venido mencionando a lo largo de este documento. A partir de las características de dicho sistema y las discusiones con el equipo de desarrollo y los dueños del sistema, se presentará las decisiones tomadas que guiaron el proceso de la creación del nuevo módulo, la migración de algunos de sus procedimientos centrales y la creación de pruebas automáticas.

2.2.1 Integración de nuevo módulo en cambio tecnológico

El sistema legado en cuestión tiene ciertas características que hacen que sea complejo su tratamiento, por un lado es de gran valor para la empresa que lo usa y tiene inmerso gran parte de la lógica del negocio de dicha compañía; por otro lado las tecnologías, ahora obsoletas, bajo las que fue implementado y su arquitectura hace que sea costoso mantenerlo y ha llevado a que se evalúe la posibilidad de migrarlo de manera definitiva.

Existen diferentes alternativas de acción frente a un sistema legado [8], pero solo dos alternativas eran aceptadas como discutibles por el área de negocio de la compañía dueña del sistema, y estas dos alternativas eran:

- Darle soporte tal y como esta.
- Migrarlo de manera gradual

No se consideraba como posibilidad retirar el sistema, ya que albergaba un proceso de suma importancia que no podía ser llevado a cabo por otra entidad ni si quiera de manera temporal.

Por su parte la primera alternativa tiene como ventaja importante el que, dado que los cambios no son radicales, hay un menor riesgo de introducción de fallas críticas en el funcionamiento del sistema, pero también tiene implicaciones negativas que representan grandes pérdidas económicas para la compañía:

- Impide que proyecten nuevos desarrollos sobre el sistema.
- Incrementa el tiempo que la compañía invierte en pruebas manuales para cada cambio que se realiza.
- Incrementa a largo plazo el tiempo de mantenimiento.

En cuanto a la segunda opción la ventaja es clara y es que se modernizaría el sistema, y las desventajas de desestabilizarlo a causa de cambios disminuían ya que se haría de manera gradual. Inicialmente se pensaba tomar módulo por módulo existente, en orden de importancia, e ir migrándolo a nuevas tecnologías, pero finalmente *se planteó una idea que implicaba menos riesgos y era que los nuevos desarrollos que se realizaran para el sistema estuvieran sobre nuevas tecnologías y serían integrados al sistema existente*. Si este proceso resultaba exitoso, entonces se podría empezar la migración gradual del sistema con cada uno de sus módulos.

2.2.1.1 Inconvenientes de la actual arquitectura del sistema

A grandes rasgos la arquitectura conceptual que el sistema actual presenta es la mostrada en la ilustración 1.

Toda la lógica de negocio del sistema se encuentra en los procedimientos pl/sql, los cuales están categorizados por paquetes que agrupaban los procedimientos para cada módulo del sistema. Por ejemplo, si el sistema tiene un módulo de usuarios entonces existe un paquete que contiene los procedimientos comunes para usuarios, registrar usuarios, consultarlos, modificarlos, etc.

El paradigma usado con los paquetes es un tipo de paradigma funcional [9], es decir, hay un conjunto de procesos sin estado que son invocados secuencialmente para realizar alguna tarea. El inconveniente con la manera en que cada procedimiento fue desarrollado es que cada procedimiento representa un proceso no reusable por otros procesos, aquí un ejemplo:

```
prc_modifyUser (userProperty1,
userProperty2, …) {

        process to modify User

}
```

```
prc_modifyUserSendingNotification
(userProperty1, userProperty2, …) {

        process to modify User

        process to send notification

}
```

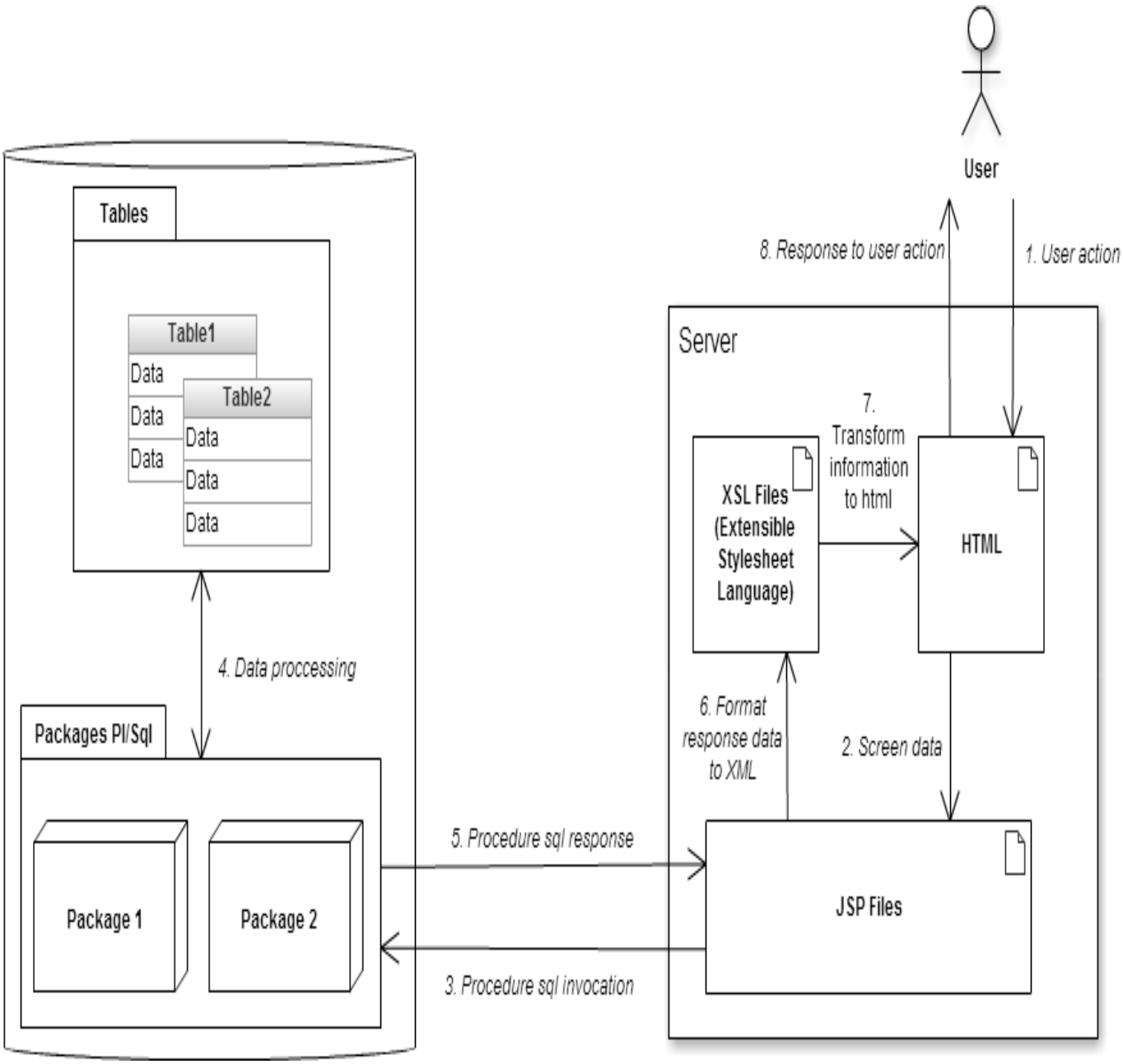


Ilustración 1 Arquitectura conceptual del sistema actual

En el procedimiento de modificar un usuario enviando una notificación se pudo haber invocado el procedimiento que únicamente modifica a un usuario y haber creado un proceso para enviar una notificación e invocarlo también, pero esto no se hizo. La duplicación de código a la que lo anterior conlleva y la dificulta para hacer uso de la reutilización es preocupante.

Otro inconveniente importante en el sistema son las hojas de estilo xsl [10] usadas para dar formato a los datos recibidos y convertirlos en html. Las ventajas que brindan el uso de esta herramienta es que se desentiende de la forma en que los datos son generados y solo brinda posibles maneras de

mostrarlos, esto permite que un esquema xml generado pueda mostrarse de diferentes maneras dependiendo de las hojas de estilo. Por ejemplo:

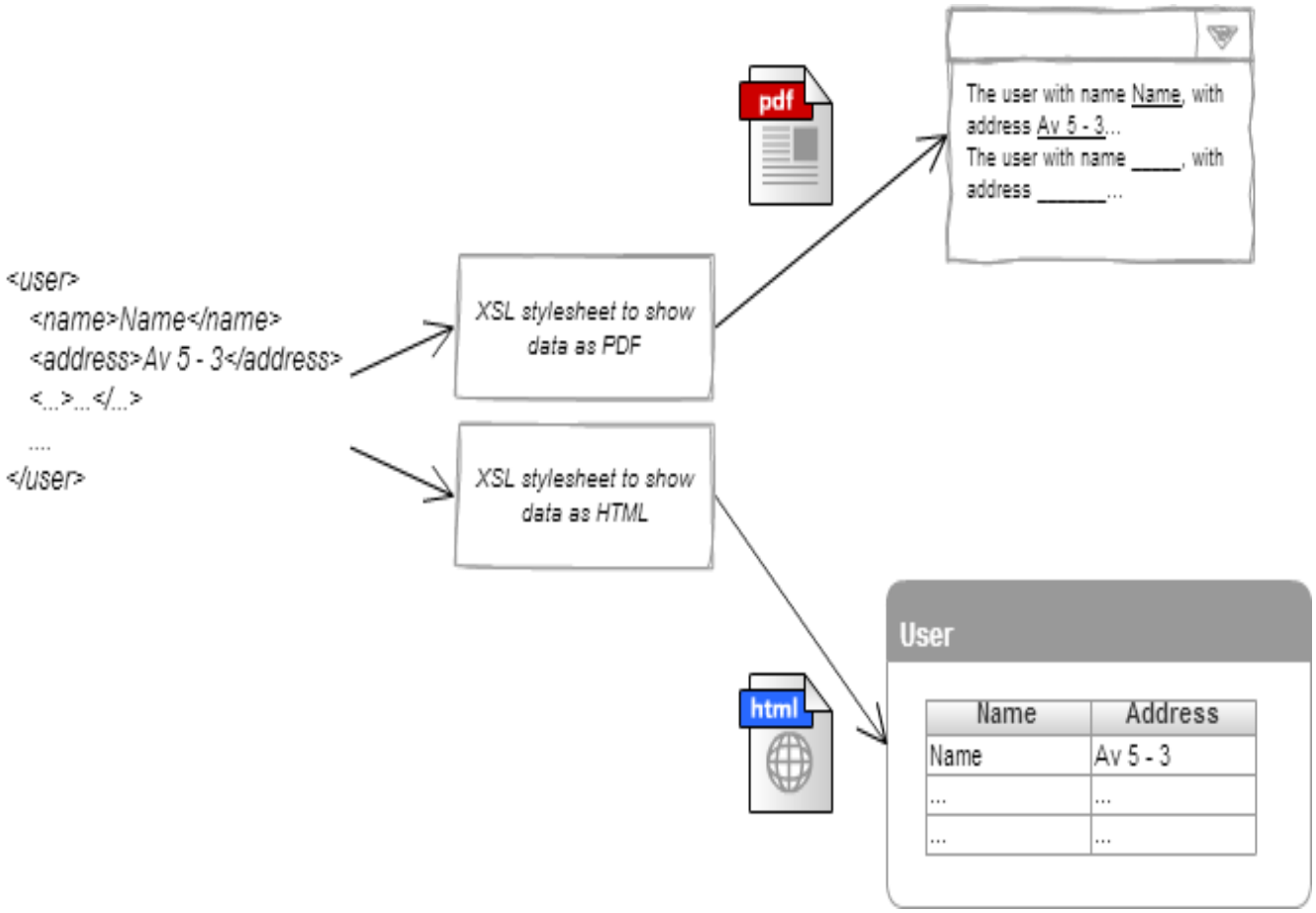


Ilustración 2 Transformaciones XSL

En cuanto a desarrollo también permite que los cambios en los esquemas no afecten el funcionamiento central del sistema ni su lógica de negocio, por ejemplo suponga que se agrega un nuevo campo al esquema xml, ¿qué partes del sistema se verían afectadas? Únicamente las hojas de estilo que dan formato a dicho esquema.

El problema con el sistema que se viene estudiando es que la lógica de negocio no era delegada únicamente las entidades que generaban los esquemas, sino que también las hojas de estilo que les daban formato contenían algo de dicha lógica. Esto hacía que lo que en principio debería ser una ventaja brindada por el uso de hojas de estilo, se convirtiera en una desventaja que duplicaba el trabajo de mantenimiento.

2.2.1.2 Marco de trabajo usado para la construcción del nuevo módulo

El nuevo módulo que se integra al sistema es una nueva herramienta solicitada por el área de negocio de la compañía propietaria. Este nuevo módulo es desarrollado usando nuevas tecnologías y una arquitectura diferente a la que el sistema legado viene manejando.

El desarrollo de este proyecto se rige bajo el marco de trabajo Scrum [11]. Habían tres roles específicos que se adaptaron de acuerdo a lo que el marco de trabajo especifica y las necesidades específicas del proyecto:

- El **Product Owner,** que en este caso era un miembro del área de negocio de la compañía propietaria, se encargaba de velar porque los intereses de la compañía propietaria cumplieran al pie de la letra, así como de responder a dudas o propuestas por parte del equipo de desarrollo.
- El otro rol era el **Scrum** Master, el cual era un miembro de Innova4j experto en la metodología Scrum; se encargaba de resolver cualquier impedimento que no permitiera que el equipo de desarrollo pudiese cumplir sus objetivos; también era el mediador entre el equipo de desarrollo y el propietario del producto.
- Finalmente el tercer rol es el de **Team Member**, este rol es el del equipo de desarrollo auto-organizado y son quienes se encargan de planificar y realizar el trabajo.

Para este proyecto el equipo de desarrollo lo compone el autor de este trabajo de grado y otro miembro de Innova4j que ya ha venido trabajando dando mantenimiento al sistema legado hace algún tiempo. El trabajo del nuevo módulo solo fue desarrollado por el autor de este documento, el compañero solo supervisó dicho trabajo a nivel técnico y realizó revisiones para cada tarea que fue completada.

El concepto clave para esta metodología es el **Sprint**, el cual es un período de tiempo en el que se llevan a cabo un conjunto de tareas específicas. Para nuestro caso duraba una semana.

La forma en que se trabajó el proyecto de la creación del nuevo módulo usando el marco de trabajo Scrum fue la siguiente:

1. Primero el área de negocio se encargó de enviar un documento que contenía los requerimientos a implantar en el nuevo módulo.
2. El equipo de desarrollo realizó un conjunto de tareas específicas para que se diera cumplimiento a los requerimientos y las estimó.
3. El Product Owner se encarga de mantener ordenadas las tareas según su prioridad.
4. Para cada sprint se programaban un conjunto de tareas de acuerdo a su prioridad e importancia. La idea era que cumplido ese conjunto de tareas, se lograra un objetivo que pudiese ser mostrado.

5. Diariamente se hacía una reunión en la que se daba seguimiento al cumplimiento del sprint: se mencionaba que se había hecho desde la última reunión, que inconvenientes se habían presentado, y que se iba a hacer hasta la siguiente reunión.

6. Al final del sprint se hacía una retrospectiva compuesta por dos partes:

    a. La primera parte era la realización de una **Demo**, que consistía en presentar al área de negocio y a todo el equipo de desarrollo lo que se había logrado en la semana.

    b. La segunda parte era establecer lo que se había **hecho bien**, y lo que hay **por mejorar**.

    La idea era constatar que los objetivos del sprint habían sido completados y lograr una mejora continua en el proceso.

2.2.1.3 Herramientas tecnológicas usadas para la construcción del nuevo módulo

Las herramientas tecnológicas que intervienen en la construcción del nuevo módulo son:

- **Maven** [12]: Es una herramienta para la gestión de proyectos. Define una manera concreta para la construcción de un proyecto basado en Java, facilitando también la especificación de las diferentes relaciones que pueden haber entre múltiples proyectos, esto mediante una característica principal llamada modelo de objeto de proyecto (POM), el cual es un archivo xml que contiene la información principal de un proyecto.

  CheckStyle [13]: Herramienta que facilita a los programadores la tarea de aplicar los estándares de programación en el desarrollo de un proyecto. Es altamente configurable, por lo que pueden aplicarse diversos estándares, o personalizarlo. En el desarrollo del nuevo módulo se usaron las reglas que vienen por defecto con la herramienta que son Sun Checks [14].

- **Spring Framework** [15]: Es un marco de trabajo para el desarrollo de aplicaciones empresariales basadas en Java. Posee útiles herramientas como inyección de dependencias, programación orientada a aspectos, Spring MVC, entre otros. Para el desarrollo del nuevo módulo se usó la inyección de dependencias y Spring MVC.

- **MyBatis** [16]: Es un marco de trabajo de persistencia que soporta SQL, procedimientos almacenados y mapeos avanzados. Fue de gran ayuda en el desarrollo del nuevo módulo, ya que, como lo especifica la documentación, permite remover la intervención de código JDBC y en cambio usar archivos de configuración mucho más mantenibles. Para el nuevo módulo los dao se definieron como interfaces java con cada método representando una operación sobre la base de datos; para cada interface Dao se crea un archivo de configuración xml que permite implementar cada método definido en la interface como una instrucción sql. La ilustración 3 muestra un ejemplo básico de su uso.

DAO como Interface Java

```java
import java.util.List;

public interface IUserDAO {

    List<User> findAllUsers() ;
}
```

Objeto de dominio Java

```java
import java.io.Serializable;

public class User implements Serializable {

    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String pName) {
        name = pName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int pAge) {
        age = pAge;
    }
}
```

Archivo de configuración XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="IUserDAO">

    <resultMap type="User" id="userResultMap">
        <result property="name" column="NAME"/>
        <result property="age" column="AGE"/>
    </resultMap>

    <select id="findAllUsers" resultMap="userResultMap">
        select * from users
    </select>

</mapper>
```

Ilustración 3 Ejemplo básico de uso de MyBatis

- **FreeMarker** [17]: FreeMarker es un motor de plantillas de texto, usado para la generación de texto que no va amarrado a un formato específico. Es software libre y corre sobre la plataforma virtual Java Virtual Machine (JVM) [3]. La razón por que se usa FreeMarker en lugar XSLT es que tiene algunas ventajas considerables que desempeñan un papel importante en el futuro de cualquier aplicación empresarial. Las diferencias técnicas son descritas en el sitio oficial de FreeMarker [18], pero las más importantes son:
    - Facilidad de uso: En lugar de tener que tratar con esquemas xml, permite usar objetos de Java. El logro principal que se alcanza con esto no es facilidad para programar, sino que el sistema pueda ser mantenido y escalado por un conjunto mayor de programadores.

- Documentación actual y centralizada: Existen muchas fuentes para consultar el XSLT que son generalmente extensas y complejas, pero la documentación oficial de FreeMarker está solo en su sitio oficial y tiene muchos ejemplos de guía que permiten agilizar el proceso de aprendizaje de este motor de plantillas de texto.

2.2.1.4 Arquitectura planteada para el nuevo módulo

Si se hubieran encontrado inconvenientes de diseño en el esquema relacional del sistema actual no se hubiese podido hacer mucho, ya que la implantación del nuevo módulo no debe afectar de ninguna manera el sistema existente. Debido a que no se encontraron inconvenientes en el esquema relacional del sistema actual, la arquitectura del nuevo módulo fue desarrollada partiendo del mismo esquema.

La arquitectura resultante es la mostrada en la Ilustración 4.

Dicha arquitectura fue evaluada y finalmente aceptada por el equipo de desarrollo ya que permite separar explícitamente las diferentes capas que componen el sistema, la capa de la vista que se encarga de mostrar la información al usuario; la capa controlador que conoce quien se debe encargar de una determinada tarea; la capa de servicio que contiene la lógica de negocio para cada módulo; y finalmente la capa de acceso a datos que se encarga únicamente de entenderse con la base de datos.

La arquitectura planteada permite realizar un producto de software de calidad y bajo tecnologías actuales, pero *no impide que se cometan los mismos errores cometidos en la creación del sistema original*, por lo que fueron tenidos muy en cuenta para no repetirlos; aquí algunos de los más frecuentes y su respectiva medida:

- *Incluir lógica de negocio en la las plantillas que generan las pantallas del sistema*: Se aseguró que las plantillas FreeMarker se encargaran únicamente de mostrar datos al usuario final sin contar con la lógica de negocio del sistema; si en las plantillas se usaban ciclos o sentencias de control, solo se hacía para mostrar los datos en la pantalla de manera adecuada, por ejemplo una lista de usuarios.

- *Incluir código Javascript de manera directa en las pantallas*: Se separó el código Javascript incluido en las pantallas en ficheros .js. Se creó un fichero Javascript para cada módulo del sistema y un conjunto de ficheros de utilidades comunes en todo el sistema, por ejemplo los que procesan fechas, validaciones de datos (como direcciones de correo bien formadas), y funciones de ordenamiento, entre otras tareas.

- *Uso de literales*: Se crearon clases que contuvieran constantes en lugar literales. Un literal en Java es un valor que aparece de manera directa en el código fuente de un programa sin ser asignado a una variable (por ejemplo: 123, “Hola”, true). La creación de constantes facilita la centralización de dichos valores y su mantenimiento se hace más sencillo. Por ejemplo en lugar de usar “Y”, que en el sistema representa un valor de verdad verdadero en la base de datos, se usó la constante YES, de modo que si en algún momento se cambia la representación del valor de verdad verdadero en la base de datos a “S”, por ejemplo, tan solo bastaría cambiar el valor de la constante YES.

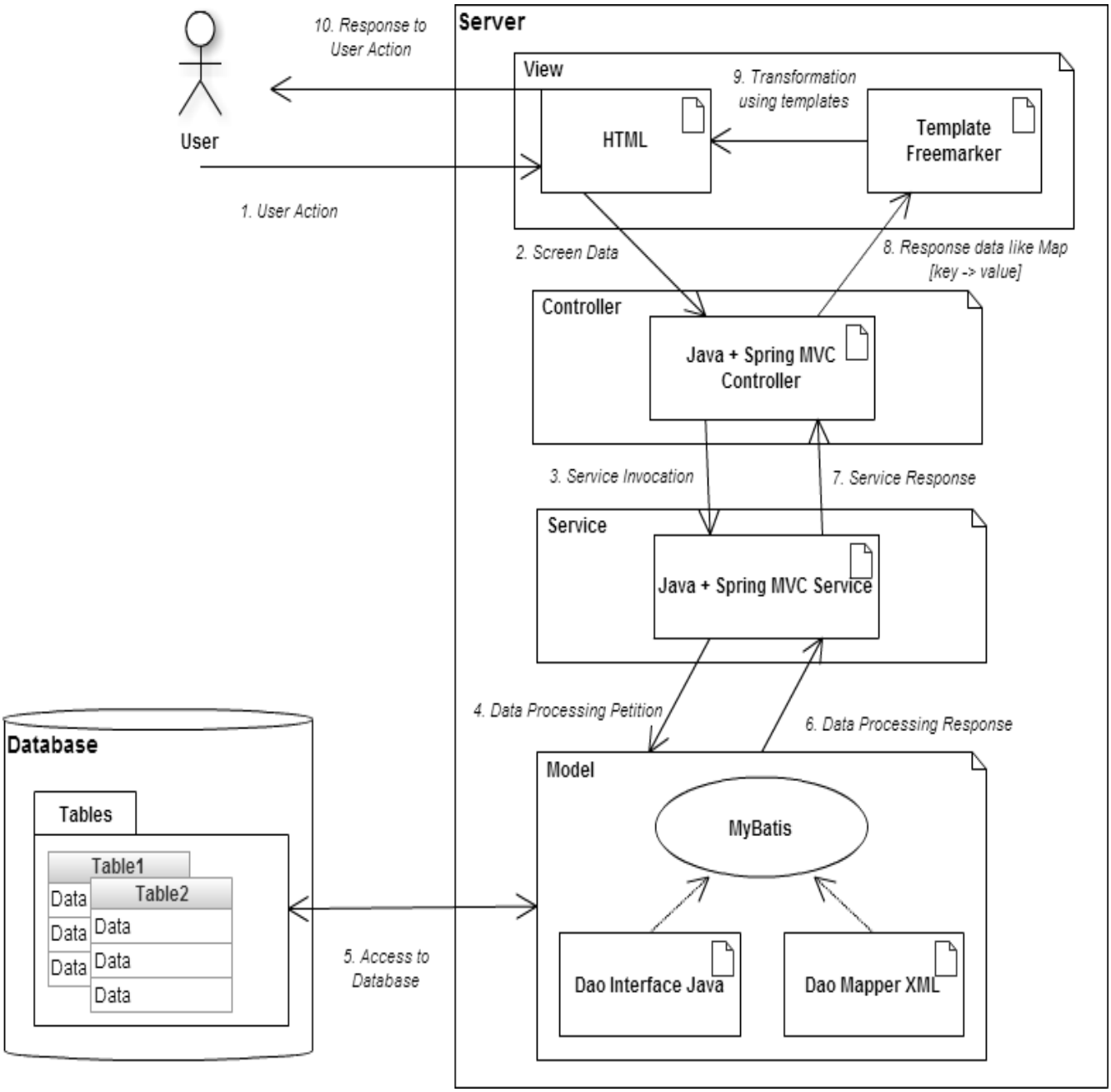


Ilustración 4 Arquitectura conceptual planteada para el nuevo módulo

El uso de Java como lenguaje orientado a objetos para la implementación de los servicios que contienen la lógica de negocio del nuevo módulo, permite organizar dichos procesos de una manera más entendible, reusable y organizada. Aspectos clave de la programación orientada a objetos como el polimorfismo o la herencia hacen de la capa de servicios una herramienta potente en lugar de un dolor de cabeza.

Esto sin mencionar que el hecho de que la lógica de negocio y el acceso a datos estén implementados en Java, facilita la construcción de pruebas de unidad automáticas con herramientas muy usadas y conocidas en el desarrollo de software, como JUnit [19], caso que no se da en el sistema original y que es el causante de la automatización de pruebas con Selenium que se expone más adelante.

2.2.1.5 Conexión entre el nuevo módulo y el sistema legado

Finalmente se presenta a continuación la forma en que se conectó el nuevo módulo, desarrollado bajo una arquitectura diferente, y el sistema legado.

La solución por la que se optó para la conexión del nuevo módulo con el sistema fue la implementación de un filtro en el sistema legado que reconociera las peticiones que se invocaban y, en caso de que fueran peticiones al nuevo módulo, se redirigiría la petición al servidor en el que el nuevo módulo estaba levantado. Se tenían entonces dos servidores, el que contenía el sistema legado (con un puerto específico, para ver que es un puerto vea [20], o para una explicación más técnica vea [21]), y el que contenía el nuevo módulo.

Para la creación de dicho filtro se usó la interface *javax.servlet.Filter* [22], que permite interceptar tanto las peticiones como las respuestas del servlet. Se muestra a continuación, de manera general, la forma como se implementó el filtro para que pudiese unirse el nuevo módulo con el sistema. Se aclara de antemano que el contenido de dicha implementación es solo lo esencial y concerniente únicamente a la parte teórica de la conexión aquí explicada, no es igual que la implementación del filtro entregada a la compañía dueña del producto.

1. Primero se especifica la implementación del filtro en el archivo web.xml, así como sus parámetros:

```
<web-app ...>
...
    <filter>
        <filter-name>controllerFilter</filter-name>
        <filter-class>ControllerFilter</filter-class>
        <init-param>
            <param-name>new-module-url</param-name>
             <param-value>localhost</param-value>
        </init-param>
        <init-param>
            <param-name>new-module-port</param-name>
            <param-value>8081</param-value>
        </init-param>
        <init-param>
            <param-name>new-module-prefix</param-name>
            <param-value>newModule</param-value>
        </init-param>
    </filter>
...
    <filter-mapping>
        <filter-name>controllerFilter</filter-name>
        <url-pattern>*.jsp</url-pattern>
    </filter-mapping>
...
</web-app>
```

Ilustración 5 Ejemplo de filtro paso 1

2. Luego se inicializan los parámetros necesarios para que el filtro funcione. Para esto se implementa el método *init(FilterConfig)* definido por la interface *Filter*. Los parámetros de configuración contenidos en el objeto *filterConfig* son los que se fijan en la especificación de los filtros del servlet en el fichero web.xml

```
import org.apache.commons.httpclient.Credentials;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpException;
import org.apache.commons.httpclient.UsernamePasswordCredentials;
import org.apache.commons.httpclient.auth.AuthScope;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.methods.PostMethod;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class ControllerFilter
    implements Filter {

    private String newModuleUrl;
    private String newModulePort;
    private String newModulePrefix;

    public void init(FilterConfig filterConfig)
    throws ServletException {
        newModuleUrl = filterConfig.getInitParameter("new-module-url");
        newModulePort = filterConfig.getInitParameter("new-module-port");
        newModulePrefix = filterConfig.getInitParameter("new-module-prefix");
    }
```

Ilustración 6 Ejemplo de filtro paso 2

3. Inicializados los parámetros del filtro, se implementa el método que será invocado cuando se realice una petición (Ilustración 7). Resumidamente lo que el método hace es que, si la petición empieza por el prefijo definido para todos los recursos del nuevo módulo, en este caso '*newPage'*, entonces redirige la petición al nuevo módulo, de lo contrario invoca el método *filterChain.doFilter*, que deja que la petición sea respondida por el sistema.

4. Finalmente se implementan los métodos que redirigen la petición al nuevo módulo (Ilustración 8).

```
    public void doFilter(ServletRequest servletRequest,ServletResponse servletResponse,
        FilterChain filterChain) throws IOException, ServletException {

        String newPageOfNewModule = "http://" + this.newModuleUrl + ":" + this.newModulePort;
        String strMethod = request.getParameter("METHOD");

        /**
        Se obtiene nombre de la pagina en la petición, por ejemplo si
        es 'google.com/applications.jsp', se obtendria 'applications.jsp'
        */
        String namePage = "";
        String requestURI = servletRequest.getRequestURI();

        int fromIndex = requestURI.lastIndexOf('/') + 1;
        int toIndex = requestURI.lastIndexOf('.');

        if (toIndex > fromIndex)
          namePage = requestURI.substring(fromIndex, requestURI.length());

        if (namePage != null) {
            if (namePage.startsWith(newModulePrefix)) {
                newPageOfNewModule = newPageOfNewModule + "/" + namePage;
                if ("POST".equals(strMethod)) {
                    executeUrlPost(session, newPageOfNewModule, session, servletResponse);
                } else if ("GET".equals(strMethod)) {
                    executeUrl(session, newPageOfNewModule, session, servletResponse);
                }
            }
            else {
                filterChain.doFilter(servletRequest, servletResponse);
            }
        }
    }
```

Ilustración 7 Ejemplo de filtro paso 3

```
    private void executeUrlPost(HttpServletRequest request, HttpSession session,
        String targetURL, ServletResponse servletResponse) {
        PostMethod postMethod = null;
        try {
            HttpClient httpClient =  new HttpClient();
            postMethod = new PostMethod(targetURL);
            Credentials defaultcreds = new UsernamePasswordCredentials(/*...*/);
            // ...
            httpClient.executeMethod(postMethod);
            String responseBody = postMethod.getResponseBodyAsString();
            servletResponse.getWriter().print(responseBody);
        }
        catch (HttpException e) {/*...*/}
        catch (IOException e) {/*...*/}
        catch (Throwable e) {/*...*/}
        finally {/*...*/}
    }
    private void executeUrl(HttpSession session, String targetURL, String username,
        String password, ServletResponse servlet) {
        GetMethod getMethod = null;
        try {
            HttpClient httpClient = new HttpClient();
            getMethod = new GetMethod(targetURL);
            Credentials defaultcreds = new UsernamePasswordCredentials(/*...*/);
            // ...
            httpClient.executeMethod(getMethod);
            String responseBody = getMethod.getResponseBodyAsString();
            servletResponse.getWriter().print(responseBody);
        }
        catch (HttpException e) {/*...*/}
        catch (IOException e) {/*...*/}
        catch (Throwable e) {/*...*/}
        finally {/*...*/}
    }
}
```

Ilustración 8 Ejemplo de filtro paso 4

Así se concluye la construcción del nuevo módulo sobre una arquitectura diferente a la del sistema legado y su integración con dicho sistema.

2.2.2 Migración de procedimientos pl/sql a Java

Para la construcción del nuevo módulo fue necesaria la migración de algunos procedimientos plsql a Java. El nuevo módulo, además de contener procesos nuevos que pueden ser implementados en Java desde cero, debe invocar procesos del sistema que ya existen; podría haberse decidido simplemente usar la herramienta MyBatis e invocar los procedimientos almacenados que se necesiten, pero debe hacerse énfasis en que uno de los objetivos de que el nuevo módulo sea implementado bajo una nueva arquitectura y usando nuevas tecnologías es evaluar que tan costoso y riesgoso sería la migración de un módulo existente del sistema.

2.2.2.1 Criterios para seleccionar la selección de los procedimientos a migrar

Por lo anterior se eligieron un conjunto de procedimientos almacenados que serían migrados a Java y usados por el nuevo módulo; los criterios para seleccionar los procedimientos que serían migrados fueron:

1. ¿Es importante el proceso que lleva a cabo el procedimiento? Si el procedimiento plsql no es de importancia para el nuevo módulo ni para el sistema entonces no vale la pena invertir tiempo en su migración.
2. ¿Es un procedimiento que no está expuesto a cambios constantes? La migración del procedimiento implica tener código duplicado porque el sistema seguirá haciendo referencia al procedimiento almacenado y el nuevo módulo invocará al proceso en Java. Cualquier cambio que se necesita hacer sería un costo doble de desarrollo, por lo que se debe asegurar de que no sea cambiado frecuentemente.
3. ¿Invoca en su proceso otros procedimientos grandes que no serán migrados? En caso de que el procedimiento a migrar invoque otros procedimientos grandes (grandes en cuanto a nivel de importancia o números de líneas de código), se tendrían que evaluar cada uno de estos, y si se decide que todos van a ser migrados entonces se migraría el procedimiento que los invoca.

2.2.2.2 Proceso de migración

La migración de estos procedimientos, a nivel técnico, podría verse como un mapeo entre código plsql y Java, pero va más allá de eso. Es una migración a nivel de proceso y no de implementación. Se deben potenciar las ventajas que ofrece el paradigma de programación orientado a objetos [9] y

específicamente el lenguaje Java [3], así como también afrontar las desventajas implicadas por el acceso a la base de datos, esto porque en los procedimientos almacenados dicho acceso era directo pero en Java debe haber un intermediario o conector.

Mejoras Introducidas

En el proceso de migración se dio un **caso muy frecuentemente en el que se usó el polimorfismo brindado por el lenguaje Java para solventarlo**; dicho caso pudo ser generalizado y explicado en el siguiente ejemplo:

Suponga la existencia de un procedimiento encargado de generar un reporte en dos posibles formatos, Word o PDF. El procedimiento recibe un texto, le da formato y genera el documento.

```
CREATE OR REPLACE PROCEDURE
PRC_GENERATE_DOCUMENT(
  P_TEXT VARCHAR2,
  P_DOCUEMNT_TYPE VARCHAR2
)
IS
BEGIN
  IF P_DOCUEMNT_TYPE = 'WORD'
  THEN
    PRC_GENERATE_DOCUMENT_WORD(P_TEXT);
  ELSIF P_DOCUEMNT_TYPE = 'PDF'
  THEN
    PRC_GENERATE_DOCUMENT_PDF(P_TEXT);
  END IF;
END;
```

```
CREATE OR REPLACE PROCEDURE
PRC_GENERATE_DOCUMENT_WORD(P_TEXT VARCHAR2)
IS
BEGIN
  -- IDENT TEXT
  -- CLEAR SPECIAL CHARACTERS
  -- APPLY HTML FORMAT

  -- CREATE EMPTY WORD DOCUMENT
  -- PUT TEXT IN EMPTY WORD DOCUEMNT
  -- GENERATE WORD
END;

CREATE OR REPLACE PROCEDURE
PRC_GENERATE_DOCUMENT_PDF(P_TEXT VARCHAR2)
IS
BEGIN
  -- IDENT TEXT
  -- CLEAR SPECIAL CHARACTERS
  -- APPLY HTML FORMAT

  -- CREATE EMPTY PDF DOCUMENT
  -- PUT TEXT IN EMPTY PDF DOCUEMNT
  -- GENERATE PDF
```

Ilustración 9 Ejemplo de polimorfismo en migración parte 1

Una de los aspectos negativos a notar de la anterior implementación es que el procedimiento principal PRC_GENERATE_DOCUMENT debe conocer los formatos que pueden ser generados y conocer también los procedimientos que los generan. Otro inconveniente importante es que entre los procedimientos PRC_GENERATE_DOCUMENT_WORD y PRC_GENERATE_DOCUMENT_PDF existe código duplicado.

La migración en Java que se realiza para este proceso es modelada por el diagrama de cases mostrado en la ilustración 10.

Implementando este modelo se logra superar los dos aspectos negativos mencionados anteriormente; ahora el proceso que genera el documento (en este caso contenido en la clase ReportManager) solo posee un objeto de tipo DocumentGenerator, el cual se encarga de generar un documento con el formato “apropiado”.

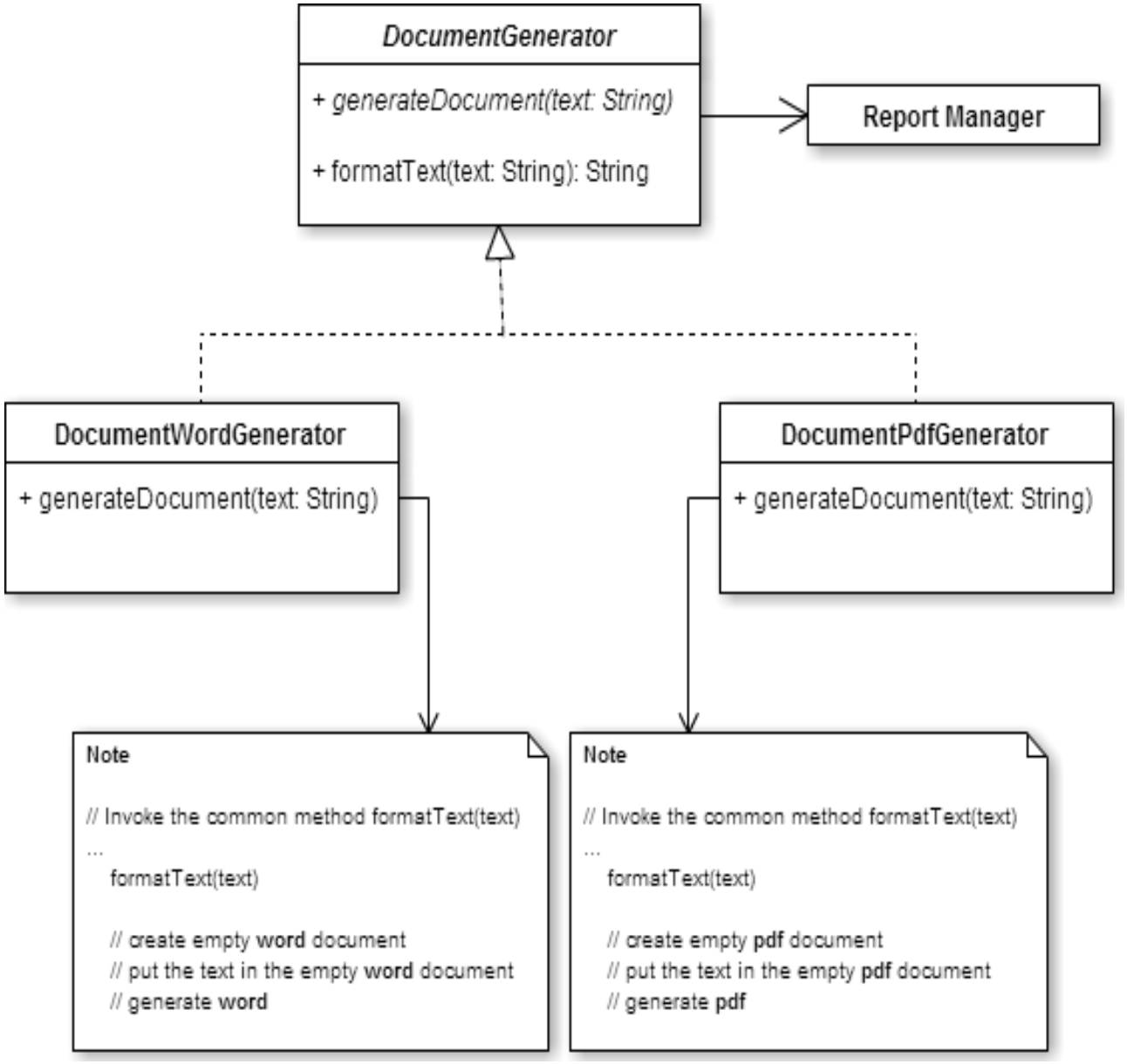


Ilustración 10 Ejemplo de polimorfismo en migración parte 2

El formato “apropiado” es el que se desea generar y este dependerá de la instancia que le sea asignada al objeto de tipo DocumentGenerator contenido en ReportManager; dependiendo de si la instancia es de tipo DocumentWordGenerator o DocumentPdfGenerator se generarán reportes en Word o Pdf respectivamente. El segundo aspecto se soluciona unificando lo que ambos procesos necesitan para evitar la duplicidad de código, en este caso se unifica en el método formatText(text) lo que ambos procesos tienen en común.

Otra mejora importante fue la **organización de los fuentes** que antes estaban agrupados por paquetes y cada paquete representaba un módulo del sistema, ahora el agrupamiento se realiza en paquetes Java y tiene más detalle, por ejemplo en lugar de crear un paquete de usuarios únicamente, se tiene un paquete de usuarios y a su vez existe un paquete de gestión de usuarios, de envío de notificaciones, de manejo de niveles de acceso, etc.

En general se realizaron mejoras relacionadas con las buenas prácticas de programación mediante la herramienta CheckStyle [13] como la documentación exhaustiva y no redundante del código fuente realizado, manejo de excepciones, visibilidad de los miembros de una clase, separación adecuada de paquetes y uso de librerías estándar.

Inconvenientes afrontados

Un inconveniente importante en los procesos migrados a Java es el **acceso a la base de datos**. Cada vez que se necesite realizar una consulta o actualización en la base de datos se debe usar un intermediario para esto, en este caso MyBatis [16]. Si se hiciera una consulta para cada cursor existente en los procedimientos plsql se hubieran creado una cantidad considerable de estas; por lo que se usó un enfoque de reutilización de consultas. La idea es que las consultas sean lo más parametrizables posible para que puedan ser de utilidad a múltiples procesos. El ejemplo que se muestra a continuación (basado en los ejemplos de la página oficial de MyBatis [23]) muestra cómo usar sql dinámico para que la cláusula WHERE pueda ser reusada por múltiples consultas que accedan a la tabla BLOG:

```
<sql id="whereBlogLike">
  <where>
    <if test="state != null">
        state = #{state}
    </if>
    <if test="title != null">
        AND title like #{title}
    </if>
    <if test="author != null and author.name != null">
        AND author_name like #{author.name}
    </if>
  </where>
</sql>
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  <include refid="whereBlogLike" />
</select>
<select id="findActiveBlogLikeCustom"
     resultType="Blog">
  SELECT state, title FROM BLOG
  <include refid="whereBlogLike" />
</select>
```

Ilustración 11 Ejemplo de SQL Dinámico con MyBatis

Como se puede observar las cláusulas WHERE de las consultas *findActiveBlogLike* y *findActiveBlogLikeCustom* son las mismas y además de esto, las condiciones del WHERE solo aplicarán para los valores que no sean nulos, por ejemplo si el parámetro *state* es nulo entonces no será tenido en cuenta como parte del filtro, de lo contrario sí.

Otro inconveniente fue el **manejo de transacciones**. En los procedimientos plsql del sistema legado que se está estudiando, el manejo de transacciones se realiza usando los comandos COMMIT, ROLLBACK y SAVEPOINT. Este ejemplo ilustra su uso básico:

```
DECLARE

BEGIN
   SAVEPOINT duplicate_found;

   UPDATE users SET id=2 WHERE name = 'Carlos';

   COMMIT;

EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
     ROLLBACK TO duplicate_found;
END;
```

Ilustración 12 Ejemplo de Manejo de transacciones PLSQL

Mediante el comando SAVEPOINT se elige un punto de referencia que indica el inicio de un proceso que puede estar expuesto a errores; en caso de que se presente una excepción, se ejecuta el comando ROLLBACK TO que reversa todo los cambios realizados a la base de datos desde el punto *duplicate_found*; en caso de que no se presenten errores, entonces se ejecuta el comando COMMIT que se encarga de guardar los cambios realizados a la base de datos.

La solución para el manejo de transacciones en Java fue usar la herramienta que ofrece ***Spring*** [15] para esto. Dicha herramienta se usa mediante anotaciones a nivel de métodos o clases y permite realizar commit o rollback basándose en las excepciones lanzadas que hereden de *RuntimeException*. Es muy configurable y su uso no requiere mayor cantidad de código.

En la ilustración 13 se presenta un ejemplo que explica su uso básico.

En el ejemplo se especifica que si el método *TaskManager.doWork*() lanza una excepción de tipo *FileNotFoundException* o que herede de ella **se hará ROLLBACK** (nótese que esto ocurre aun cuando la excepción *FileNotFoundException* no hereda de *RuntimeException* solo por haberla especificado en los parámetros de la anotación).

```
public class TaskManager {

    @Autowired
    private Tool tool;

    @Transactional(
        rollbackFor={FileNotFoundException.class},
        noRollbackFor={ArithmeticException.class}
        )
    public void doWork() {
        tool.doSomethingDangerous();
    }
}
```

Ilustración 13 Ejemplo de uso de Transacciones en Spring

Y especifica que si se genera una excepción de tipo *ArithmeticException* o que herede de ella, **no se hará ROLLBACK**.

De esta manera se llevó a cabo el proceso de migración de algunos de los procedimientos almacenados que contienen procesos que son parte del sistema y que también deben ser llevados a cabo por el nuevo módulo.

2.2.3 Automatización de pruebas para el sistema legado

Se presenta a continuación el proceso seguido para la construcción de un proyecto que contiene un conjunto de pruebas automáticas para el sistema legado, su arquitectura base y las herramientas tecnológicas que lo soportan. El proyecto no pretende contener todas las pruebas para el sistema legado, tan solo contiene pruebas que servirán de base para construir cada vez más y más pruebas.

2.2.3.1 Situación del sistema legado

Antes de la construcción del proyecto de automatización de pruebas no existían pruebas automáticas que se aseguraran del correcto comportamiento de las funcionalidades del sistema legado. El sistema seguía recibiendo mantenimiento y cada vez que se realizaban cambios las consecuencias eran problemáticas, porque surgían errores causados por dichos cambios. Tanto era el riesgo que el área de negocio debía ejecutar un plan de pruebas manual para todo el sistema por cada cambio realizado.

El hecho de que sea un sistema legado acota las maneras en que las pruebas pueden ser automatizadas, ya que la lógica del sistema no está implementada como un conjunto procesos

separables y probables de manera individual. Por tal motivo, la propuesta de automatización de pruebas que más adelante se presenta trata de pruebas que se entienden directamente con la aplicación, en otras palabras simulan ser el usuario final (vea Selenium [24]).

La justificación de la automatización de las pruebas, además de lo anterior, obedece al objetivo a largo plazo que se ha fijado para el sistema y es una migración completa. La automatización de las pruebas no es más que un tipo de *blindaje* para el sistema que asegura su correcto comportamiento aún ante cambios masivos que puedan llegar a desestabilizarlo.

2.2.3.2 Planteamiento de la propuesta

Se decidió junto al equipo de desarrollo que las pruebas desarrolladas debían ser de caja negra, es decir, que prueben la funcionalidad del sistema desde el punto de vista del usuario, sin probar los componentes del sistema de manera interna.

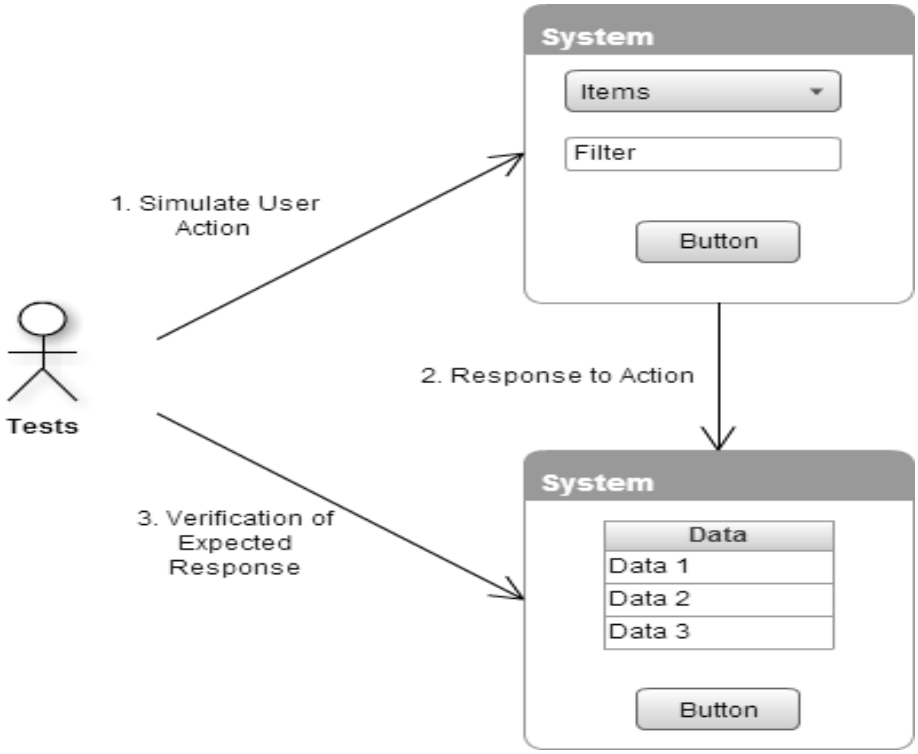


Ilustración 14 Esquema conceptual de pruebas automáticas

La razón por la que se decidió que las pruebas fueran de caja negra, es que el sistema podría ser migrado a nuevas tecnologías y aún bajo este contexto las pruebas seguirían teniendo validez ya que la capa de presentación del sistema y su lógica de negocio serían las mismas.

Herramientas tecnológicas

Las herramientas tecnológicas seleccionadas ofrecen la posibilidad de crear las pruebas de manera sencilla usando una interfaz gráfica de usuario, para posteriormente exportarlas a un conjunto de pruebas en Java que pueden ser ejecutadas de manera automática.

- **Selenium** [24]**:** Entorno con las características necesarias para la ejecución de pruebas automáticas en aplicaciones basadas en la web. Una de las características más importantes de esta herramienta es que no requiere conocimientos de algún lenguaje específico por parte de quien la usa.

- **TestNG** [25]**:** Marco de trabajo para la realización múltiples tipos de pruebas de software. Es configurable, de fácil manejo y con muchas funcionalidades. Entre las ventajas que tiene el uso de esta herramienta destacan la parametrización de las pruebas, la dependencia entre pruebas y la opción de agrupamiento de pruebas.

Proceso de creación de pruebas

Para la creación de pruebas se usa un plugin en Firefox llamado Selenium IDE. Este se encarga de grabar y exportar cada prueba realizada. El paso a paso se muestra creando una prueba que inicia sesión en un sistema con las credenciales “jhonacd” y “1234”, y se dirige al módulo “Productos”.

1. Se abre Selenium IDE y el navegador Firefox, como muestra la ilustración 15.

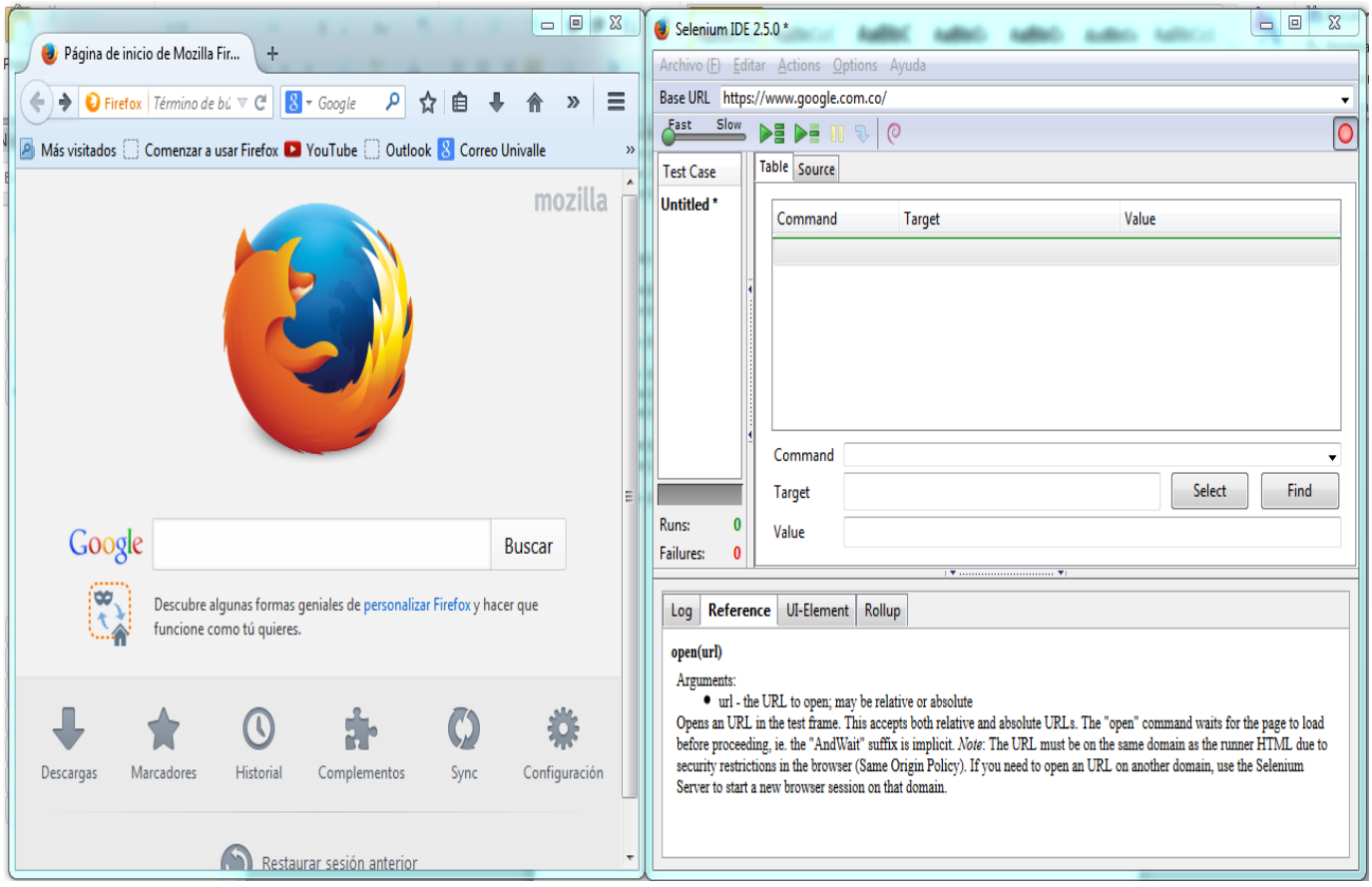


Ilustración 15 Paso 1 para la creación de una prueba.

2. Se realizan la acción concreta a la aplicación de manera manual mediante el navegador Firefox, tomando el rol del usuario final. Esta acción es grabada de manera automática por Selenium IDE, como se muestra en la ilustración 16.

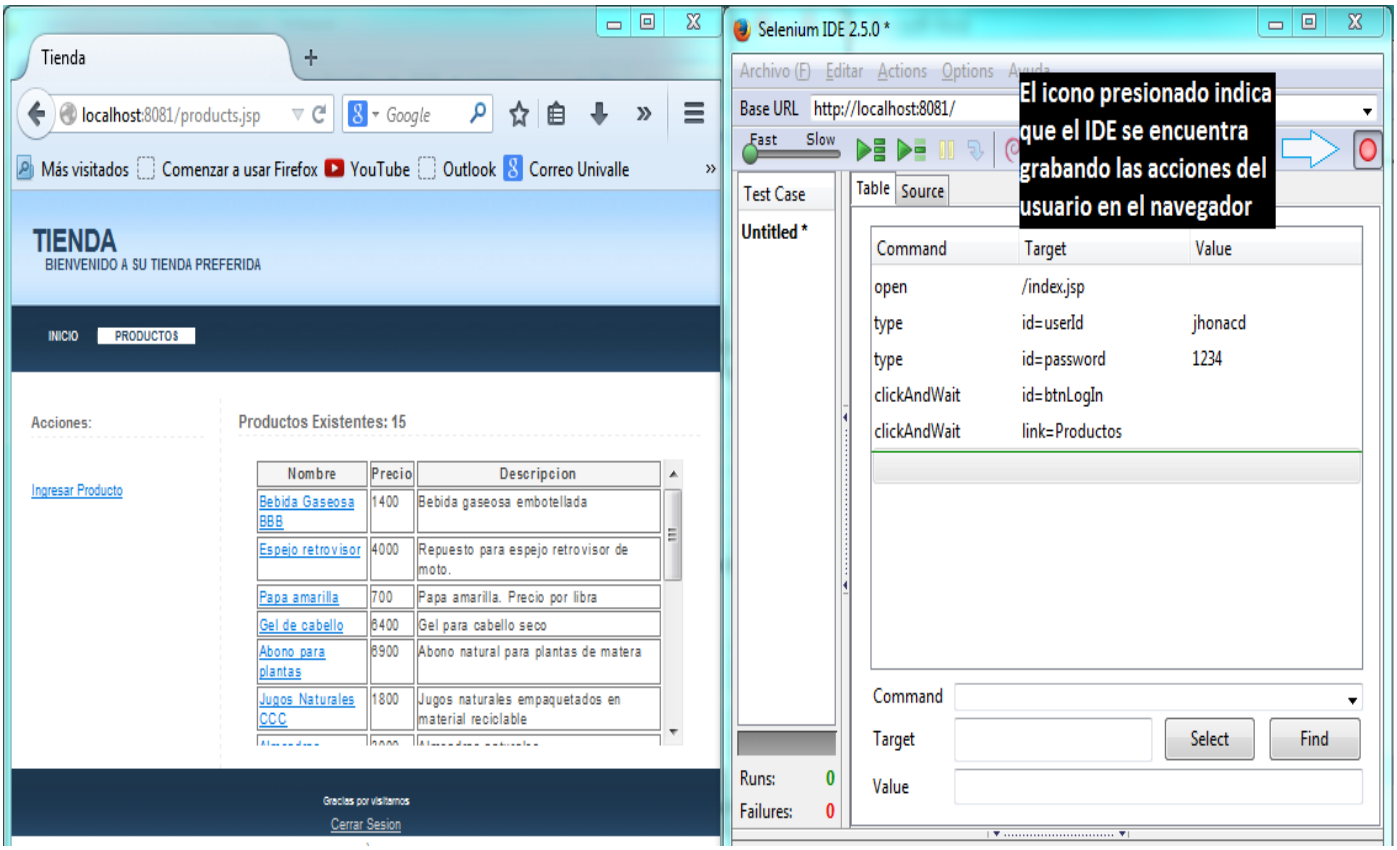


Ilustración 16 Paso 2 para la creación de una prueba.

3. Mediante Selenium IDE, se exporta la prueba realizada a un archivo fuente (Java en este caso).

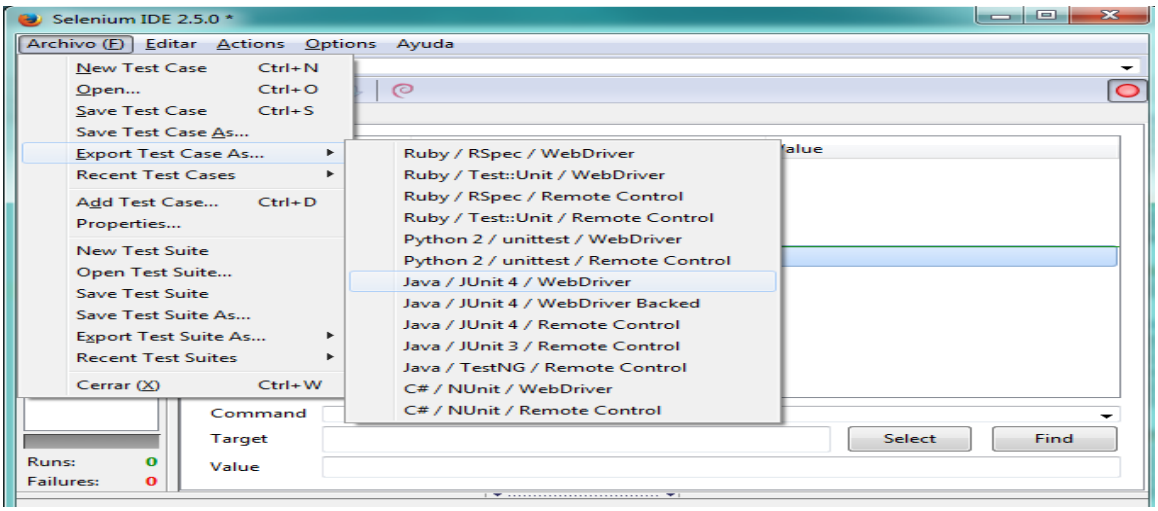


Ilustración 17 Paso 3 para la creación de una prueba.

4. Finalmente el resultado de la exportación es una clase de prueba Java tal y como se observa en la ilustración 18. Se construye una clase que contenga esa prueba usando TestNG. La construcción de la clase de prueba debe usar constantes en lugar de literales para lograr la creación de un banco de constantes que permitan el reúso de las mismas.

```
public class Test {

  private WebDriver driver;
  private String baseUrl;

  @Before
  public void setUp() throws Exception {
    driver = new FirefoxDriver();
    baseUrl = "http://localhost:8081/";
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
  }

  @Test
  public void test() throws Exception {
    driver.get(baseUrl + "/index.jsp");
    driver.findElement(By.id("userId")).clear();
    driver.findElement(By.id("userId")).sendKeys("jhonacd");
    driver.findElement(By.id("password")).clear();
    driver.findElement(By.id("password")).sendKeys("1234");
    driver.findElement(By.id("btnLogIn")).click();
    driver.findElement(By.linkText("Productos")).click();
  }

  @After
  public void tearDown() throws Exception {

  private boolean isElementPresent(By by) {

  private boolean isAlertPresent() {

  private String closeAlertAndGetItsText() {
}
```

Ilustración 18 Paso 4 para la creación de una prueba.

Arquitectura de proyecto de pruebas

La arquitectura del proyecto de pruebas es una arquitectura que pretende desacoplar las pruebas automáticas escritas en Java de la interacción con la interfaz gráfica usando Selenium. Es decir, si por ejemplo la prueba es crear un usuario en el sistema y luego consultarlo para verificar los datos, la prueba debe únicamente especificar los datos del usuario e invocar su creación, luego consultarlo y realizar las respectivas verificaciones, todo esto sin tener que conocer cómo se crea un usuario en el sistema o como se consulta. Quien sabe cómo interactuar con el sistema debe ser una clase de utilidad que use Selenium para realizar sus tareas.

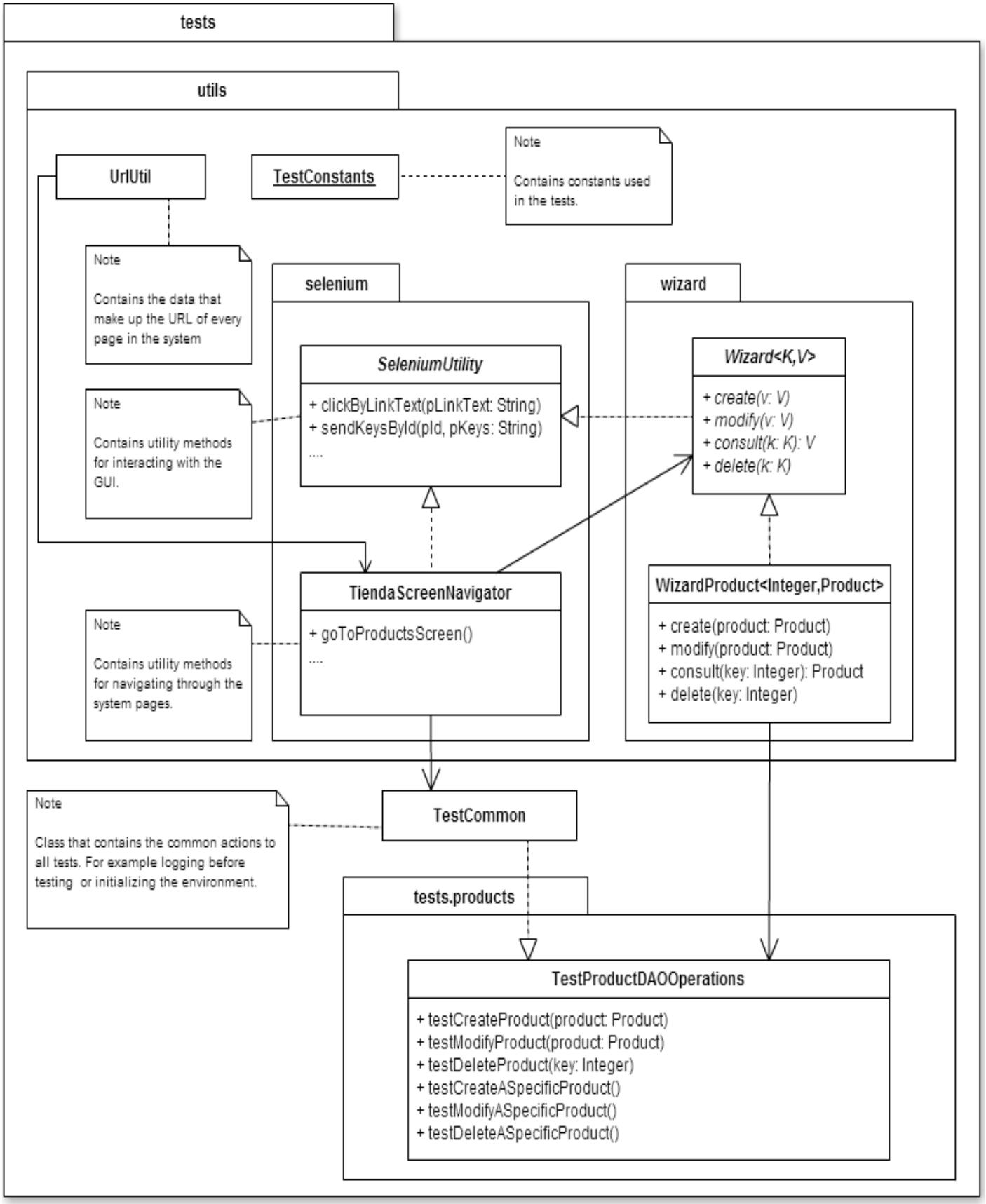


Ilustración 19 Diagrama de clases de pruebas automáticas

El diagrama de clases mostrado en la Ilustración 19 es un ejemplo que permite ilustrar la organización conceptual del proyecto de pruebas del sistema legado y será usado como caso práctico para el estudio de dicho proyecto de aquí en adelante. Para este ejemplo suponga el siguiente escenario:

El sistema representa una Tienda y permite la gestión de productos, esto es su creación, consulta, modificación y eliminación.

La división que se muestra en el diagrama de clases puede ser explicada de la siguiente manera:

- ***SeleniumUtility***: Es una clase que contiene un conjunto de métodos que permiten interactuar con la interfaz gráfica de usuario. Selenium ofrece para Java una clase llamada *WebDriver* que es la que realmente soporta la interacción con la interfaz gráfica, pero su uso puede llegar a ser no tan intuitivo; por dicho motivo la clase *SeleniumUtility* oculta los detalles de implementación de *WebDriver* y ofrece métodos intuitivos y fáciles de invocar.

  Por ejemplo, para realizar la acción de presionar un botón en una página web con una instancia de la clase *WebDriver* se haría de la siguiente manera:

  ```
  webDriver.findElement(By.id("Id_element")).click();
  ```

  Pero con el uso de la clase *SeleniumUtility* se invoca solamente el método de interés pasando como parámetro solo lo necesario, en este caso el identificador del elemento:

  ```
  seleniumUtility.clickById("Id_element");
  ```

  Como es de suponerse, la implementación del método *clickById()* no es compleja, pero su existencia si es bastante útil para el desarrollo del proyecto.

- ***TiendaScreenNavigator***: Clase de utilidad que contiene los métodos necesarios para navegar a través de las diferentes pantallas que componen el sistema. La clase extiende de *SeleniumUtility* por ser precisamente una clase de utilidad que desentiende las pruebas del uso de Selenium. Usa los métodos de su clase padre pero para un propósito más específico, y es poder ir de pantalla en pantalla a través de la invocación de métodos muy puntuales y parametrizables; por ejemplo suponga que en el sistema que representa la tienda se quiere ir a la pantalla de resultados de búsqueda de productos, y además quiere que se busque un producto específico, lo único que tendría que hacer es la siguiente invocación:

  ```
  int codeOfProduct = 123;

  screenNavigator. goToProductsScreenByProductCode(codeOfProduct);
  ```

- ***Wizard<K,V>***: Es una interface Java que define los métodos que debe implementar una clase *Wizard*. Una clase *Wizard* es una clase que conoce como crear, consultar, modificar y eliminar una determinada entidad del sistema. Por ejemplo la clase *WizardProduct* implementa las diferentes operaciones definidas por la interface *Wizard* para objetos de tipo *Product*. A partir de un objeto de tipo *Product* el método *create(p: Product)* conoce a que pantallas del sistema debe dirigirse y que campos llenar con la información del producto para poder crearlo. Las

clases genéricas ***K*** y ***V*** son la clase que representa el identificador de la entidad y la clase que representa la entidad en sí, respectivamente.

Por ejemplo la clase *WizardProduct* especifica las clases específicas *java.lang.Integer*, que es la clase que representa el código de un determinado producto, y *Product*, que representa un producto (*WizardProduct* <*Integer,Product*>).

- ***TestCommon***: Es una clase que define todos los métodos comunes de las pruebas. También ejerce control en la ejecución de todas las pruebas, como encargarse de preparar el ambiente antes de ejecutar un suite de pruebas, o iniciar sesión antes de cada grupo de pruebas, o ir a la pantalla de inicio del sistema después de cada prueba individual.

- ***TestConstants***: Contiene un banco constantes nombradas de tal manera que puedan ser usadas a través de todo el sistema. Con el uso de Selenium para navegar e interactuar con la parte gráfica del sistema se debe intervenir en todo momento con identificadores de etiquetas html, nombres, textos de hipervínculos, textos en cajas de opciones (combos), etc. Así que se centralizaron dichos valores como constantes a causa de su frecuente a parición en las pantallas del sistema.

- ***TestProductDAOOperations***: Finalmente se tienen las clases centrales del proyecto, y son las que representan las pruebas al sistema; para el ejemplo, la clase *TestProductDAOOperations* contiene métodos de prueba que se encargan de verificar el correcto comportamiento del sistema al crear, modificar y eliminar un producto.

  Se hace notorio el desacoplamiento de las pruebas con Selenium ya que solo se entienden objetos de dominio. Por ejemplo la estructura de la prueba *testCreateProduct(pProduct: Product)* sería la siguiente:

```
.. .. testCreateProduct (Product pProduct) {

.. ..

     wizardProduct.create(pProduct);

     Product productObtained =
     wizardProduct.consult(pProduct.getCode());

     assertEquals(productObtained.getName(), pProduct.getName());

.. ..

}
```

Proceso de ejecución de pruebas

Una vez creadas las pruebas individuales y reunidas en un proyecto de pruebas Maven [12] con la arquitectura anterior, se debe configurar la ejecución de éstas pruebas haciendo uso las facilidades provistas por TestNG [25].

El siguiente ejemplo muestra la creación de un archivo xml que representa el plan de ejecución de un conjunto de pruebas ya creadas (vea ilustración 20). Este archivo se nombra para este ejemplo como testProducts.xml y es interpretado y ejecutado por TestNG [25].

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="false" preserve-order="true">
    <test name="TestProductsBasicOperations" enabled="true" preserve-order="true">
        <classes>
            <class name="com.tienda.tests.products.TestDAOProduct">
                <methods>
                    <include name="testCreateProduct" />
                    <include name="testModifyProduct" />
                    <include name="testDeleteProduct" />
                </methods>
            </class>
            <class name="com.tienda.tests.products.TestChangeQualityGrade">
                <methods>
                    <include name="testProductChangeQualityGrade" />
                </methods>
            </class>
        </classes>
    </test> <!-- TestProductsBasicOperations -->
</suite> <!-- Suite -->
```

Ilustración 20 Configuración para la ejecución de un conjunto de pruebas.

Habiendo creado este archivo xml en el proyecto, puede ser ejecutado ahora de manera individual, corriendo por ende las siguientes pruebas (en este orden):

- `TestDAOProduct.testCreateProduct()`
- `TestDAOProduct.testModifyProduct()`
- `TestDAOProduct.testDeleteProduct()`
- `TestChangeQualityGrade.testProductChangeQualityGrade()`

Para evitar que la ejecución de las pruebas dependa de la ejecución del archivo anterior, se agrega el siguiente plugin a la configuración Maven [12] del proyecto de pruebas:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.tienda.tests</groupId>
    <artifactId>tests</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>tests</name>
    <url>http://maven.apache.org</url>

    <properties>...

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.16</version>
                <configuration>
                    <suiteXmlFiles>
                        <suiteXmlFile>src/test/resources/testProducts.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>...
</project>
```

Ilustración 21 Configuración de ejecución para el proyecto de pruebas

Con el proyecto de pruebas configurado de la anterior manera, cada vez que el proyecto mismo se ejecute o se instale como un proyecto Maven [12], las pruebas establecidas en el plan de ejecución testProducts.xml serán ejecutadas. Note que otros planes de ejecución pueden ser agregados a la configuración del proyecto usando la etiqueta <suiteXmlFile></suiteXmlFile>.

La arquitectura mostrada definitivamente permitió que a medida que el proyecto de pruebas crece, la creación de pruebas para el sistema concierna únicamente al establecimiento de su comportamiento y su implementación a un alto nivel.

2.2.3.3 Inconveniente en la ejecución de la propuesta y su solución

En la ejecución de la propuesta de automatización de pruebas se presentó un inconveniente no previsto durante el diseño del proyecto de pruebas, este inconveniente consistía en que habían entidades en el sistema cuyos parámetros de creación influían en propiedades asignadas por el sistema a dichas entidades de manera automática. Para ejemplificar el inconveniente suponga el siguiente escenario:

En el sistema descrito anteriormente que representa una tienda y que permite gestionar productos, cada producto puede o no tener cinco características específicas:

1. ***Es una materia prima.***
2. ***Es un producto semi-elaborado.***
3. ***Es un repuesto***
4. ***El producto tiene garantía.***
5. ***El producto esta empacado en un material reciclable***

Dependiendo de la combinación de las anteriores cinco características, el sistema asigna un grado de calidad al producto.

El proceso que usa el sistema para asignar propiedades a la entidad creada, o para el caso del ejemplo asignar un grado de calidad al producto, es un proceso determinista y puede ser replicado por el proyecto de pruebas; pero es precisamente ese proceso el que se quiere probar. No tendría mucho sentido replicar un proceso que se está probando para verificar que funcione, ya que así se le estaría introduciendo en las pruebas otro proceso vulnerable a errores y tendría que ser a su vez probado. Lo que se debería hacer es que, antes ciertas entradas, el sistema debería responder de una única manera.

Solo para el caso de ejemplo, ¿cuantas posibles combinaciones de las cinco características podrían realizarse?, y ¿cómo saber cuál es el grado de calidad que el sistema asigna para cada combinación sin replicar el proceso?

Para el sistema legado que aquí se estudia el caso era más complicado, ya que se tenían muchos más factores que intervenían en el resultado del sistema, y no se podía saber a priori que resultado sería a menos que se calculara manualmente. Por esto se diseñó una solución que permite tener entradas para las pruebas automáticas y que las pruebas puedan saber de antemano el resultado que el sistema debería arrojar.

La solución consiste consultar y almacenar todas las entidades que se han creado hasta el momento en el sistema, de esa manera dichas entidades pueden ser usadas como casos de prueba. Para el ejemplo de la tienda, se coleccionarían todos los productos (o una cantidad razonable en proporción a las posibles entradas) que se han creado en el sistema hasta el momento y, partiendo de que el grado de calidad que se les fue asignado por parte del sistema es correcto, se crea un caso de

prueba que, para cada producto coleccionado, cree otro producto por medio del sistema con las mismas características y verifica que el grado de calidad generado sea el mismo que el del producto coleccionado.

Para la implementación de esta solución al inconveniente presentado se definió que la manera en que se iban a guardar la colección de productos era en archivos xml, y cada conjunto de productos sería una plantilla. El siguiente diagrama ilustra la manera en que las pruebas van a ser “alimentadas” con estas plantillas generadas:

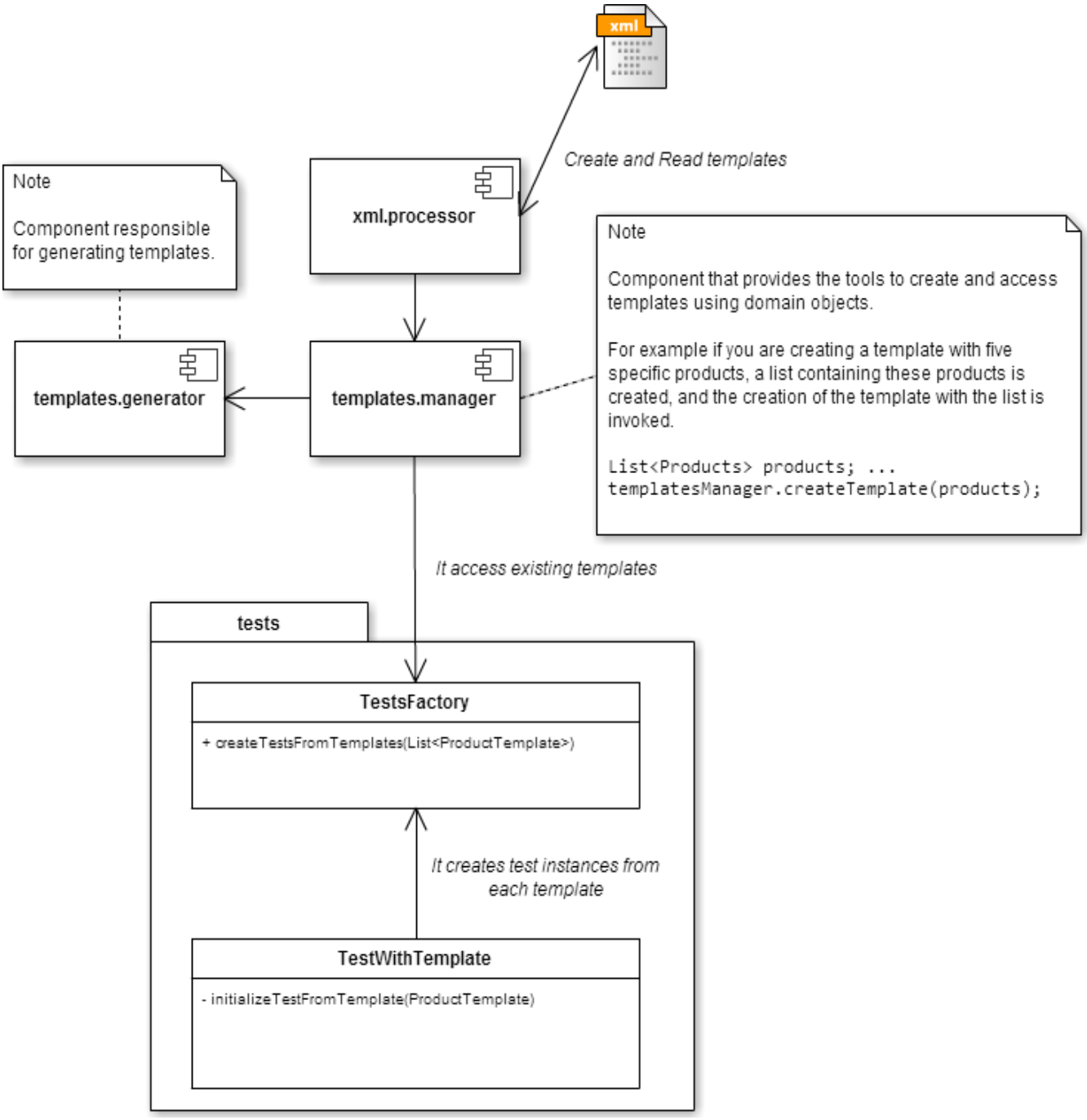


Ilustración 22 Uso de plantillas como entradas para las pruebas

A grandes rasgos el rol de cada uno de los componentes mostrados en el diagrama anterior consiste en lo siguiente:

1. El componente de más bajo nivel y que se entiende directamente con los archivos XML es ***xml.processor***. Esto usa la librería JAXB [26] para realizar sus tareas, ocultando de esta manera los detalles de uso de dicha librería. En las clases provistas por este componente están los métodos necesarios para acceder a un archivo XML y convertirlo a un objeto Java (cualquier objeto cuya clase este configurada con las anotaciones de JAXB [26] para ser transformada a XML).

2. El componente intermediario entre las plantillas XML y los objetos de dominio que representan los productos es ***templates.manager***, este se entiende con el componente ***xml.processor*** y contiene el dominio de las entidades a transformar, en este ejemplo la clase ***Product***.

3. El componente ***templates.generator*** se encarga de extraer del sistema la colección de entidades, en este ejemplo productos, y las guarda como archivos XML usando el componente ***templates.manager***.

4. La clase ***TestFactory*** usa el componente ***templates.manager*** para acceder a las plantillas ya generadas y obtener un conjunto de productos con los que probar.

5. Para cada producto, la clase ***TestFactory*** crea instancias de la prueba ***TestWithTemplate***, que a partir de una plantilla, ejecuta un caso completo de prueba para verificar que lo que genera el sistema coincida con lo que la plantilla indica. Esta prueba consiste en crear un producto con las mismas características del producto proveniente de la plantilla y una vez creado verificar que el sistema le haya asignado exactamente el mismo grado de calidad que al producto de la plantilla.

La herramienta que facilitó la generación y el acceso a las plantillas XML fue la usada por el componente ***xml.processor*** y es JAXB [26], herramienta que permite representar las clases de Java como esquemas xml por medio de anotaciones. La parametrización de las pruebas para que pudiesen usar plantillas fue posible gracias a la herramienta TestNG [25].

La ilustración 23 muestra la forma en que se debe construir la clase ***Product*** para que sus instancias puedan ser transformadas, mediante la herramienta JAXB, a un esquema XML. De esta manera, una instancia cualquiera de la clase ***Product*** con código de producto 82, nombre de producto *“Bebida Gaseosa BBB”* y descripción *“Bebida gaseosa embotellada”* es transformada a un XML como el que se muestra en la ilustración 24.

También el esquema XML transformado puede ser leído y convertido nuevamente a una instancia de clase Java, con la seguridad de que la información contenida en dicha instancia va a ser la misma que la que originalmente se creó (código de producto 82, nombre de producto *“Bebida Gaseosa BBB”* y descripción *“Bebida gaseosa embotellada”*).

```java
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;

public class Product implements Serializable {

    public int getCodProduct() {
        return codProduct;
    }

    @XmlAttribute
    public void setCodProduct(int pCodProduct) {
        codProduct = pCodProduct;
    }

    public String getName() {
        return name;
    }

    @XmlAttribute
    public void setName(String pName) {
        name = pName;
    }

    public String getDescription() {
        return description;
    }

    @XmlElement
    public void setDescription(String pDescription) {
        description = pDescription;
    }
}
```

Ilustración 23 Uso de las anotaciones de JAXB para transformaciones a de Java a XML

```xml
<products codProduct="82" name="Bebida Gaseosa BBB">
    <description>Bebida gaseosa embotellada</description>
</products>
```

Ilustración 24 Xml resultante de la transformación de un producto específico

En cuanto a la parametrización de las pruebas gracias a TestNG se puede decir que es muy útil, ya que no se interviene en nada para que la misma prueba sea ejecutada con diferentes parámetros, si no que basta con usar la anotación provista por TestNG para estos casos: @Factory. Dicha anotación es puesta en la definición de un método que retorna una lista de objetos de tipo Object. Cada elemento de la lista es una prueba lista para ser encolada (en caso de que no se permita la ejecución de pruebas de manera paralela) y posteriormente ejecutada. En el caso de la tienda, cada prueba retornada por el método de fábrica de pruebas, es una instancia de la clase ***TestWithTemplate*** y se encarga de validar que el sistema asigne un mismo grado de calidad a dos productos con las mismas características.

Aquí se ilustra cómo sería el cuerpo de la clase encargada de fabricar las pruebas a partir de plantillas de productos:

```
public final class TestFactory {

    /**
     * Crea y retorna las instancias de la prueba que crea productos
     * a partir de plantillas y verifica la igualdad del grado de
     * calidad generado por el sistema con el grado de calidad
     * del producto.
     *
     * @param pTemplateId Identificador de la plantilla a usar
     */
    @Factory
    @Parameters(value = { "id"})
    public Object[] createInstances(final String pTemplateId) {

        List<Product> products = templatesManager
                                    .getProductsFromTemplateByID(
                                        pTemplateId);

        Object[] result = null;
        if (products != null) {

            result = new Object[products.size()];
            for (int i = 0; i < products.length; i++) {
                result[i] = new TestWithTemplate(
                            products.get(i));
            }
        }
        return result;
    }
}
```

Ilustración 25 Método encargado de fabricar clases de prueba

Con la arquitectura del proyecto de pruebas planteada inicialmente y la implantación de la solución del problema presentado en la ejecución de la propuesta se pudo dar culminación al objetivo de crear un conjunto de pruebas que sirvieran como punto de partida para que futuras pruebas fueran creadas y cada vez con más nivel de abstracción.

También mediante el proyecto de automatización de pruebas se pudieron realizar un conjunto de pruebas de aceptación para el nuevo módulo definidas por el área de negocio de manera escrita y transformadas a casos de prueba mediante Selenium [24]. Esto último fue uno de los factores decisivos para que el área de negocio de la compañía propietaria del sistema legado creyera en la rigurosidad y el cubrimiento provisto por las pruebas automáticas, ya que al principio se mostraron algo incrédulos con que unas pruebas que solo pudieran interactuar con la interface gráfica pudieran probar a fondo el funcionamiento del sistema.

3 CONCLUSIONES

Después de ser parte en todo sentido del proceso que fue descrito en este documento de manera teórica, soy consciente de los grandes retos a los que conlleva el mantenimiento de un sistema legado. Enfrentar dichos retos es un trabajo que requiere bases teóricas muy fuertes para poder obtener finalmente buenos resultados. El camino seguido con el sistema legado que aquí se estudia no es más que una respuesta a las necesidades de la compañía propietaria de extender las funcionalidades de su sistema, pero minimizando las probabilidades de vulnerarlo. Habían en esencia tres aspectos centrales que se relacionaban entre sí: la necesidad de un nuevo módulo, la automatización de pruebas para el sistema completo y el análisis de que tan riesgoso sería pensar en una migración completa del sistema de manera gradual.

Dichos aspectos fueron atacados directamente, la automatización del sistema fue una especie de vacuna para que estuviera preparado ante los cambios que sufre de manera frecuente. La adición del nuevo módulo fue posible gracias a que el sistema estaba protegido, y además de que este nuevo módulo supliera las necesidades de la compañía propietaria permitió tener un acercamiento a lo que sería la migración completa del sistema legado.

A partir de la experiencia adquirida en este proceso, puedo concluir un proceso a seguir, a nivel conceptual, cuando se requiera dar mantenimiento a un sistema legado o agregarle un nuevo módulo:

1. Cree pruebas de aceptación que validen el funcionamiento del sistema desde el punto de vista del cliente. Para esto puede usar herramientas de automatización de navegación como Selenium [24]. No es necesario en todos los casos crear un plan de pruebas para todo el sistema antes de hacerle mantenimiento, puede hacer las pruebas en función de las partes del sistema que van a verse afectadas. Pero es importante que cada cambio realizado sea cubierto por una prueba automática.

2. Realice el mantenimiento del sistema teniendo siempre en mente un cambio tecnológico o migración a futuro; haga consciente al dueño del software de la necesidad existente de migrar el sistema con argumentos válidos. Cada sistema tiene sus problemas particulares.

3. En caso de que no sea viable una migración o que simplemente sea rechazada por el dueño del software, no extienda las malas prácticas que usualmente tienen los sistemas legados, por el contrario haga el mejor esfuerzo por aplicar sus criterios de calidad de software, aun cuando tenga que trabajar sobre tecnologías obsoletas.

La respuesta a la pregunta ¿es viable dar inicio a la migración completa del sistema de manera gradual migrando módulo por módulo?, sería entonces sí. Sí, pero con todas las precauciones que la situación requiere, esto es mediante la creación de pruebas que validen el correcto funcionamiento de cada módulo a medida que este va a siendo migrado para dar la tranquilidad a la compañía de que en ningún momento el sistema va a estar expuesto a errores críticos, sino que por el contrario va a ser parte de un proceso de cambio muy cuidados soportado por pruebas en amplitud y profundidad. Siendo amplitud la cantidad de pruebas que prueban determinada funcionalidad y profundidad el nivel de detalle de cada prueba.

En cuanto a las herramientas tecnológicas que se usaron a largo de este proceso se puede concluir que fueron de mucha utilidad, ya dieron solución a todos y cada uno de los problemas presentados; La conexión entre el nuevo módulo y el sistema legado (*javax.servlet.Filter* [22]), el manejo de transacciones (*Spring* [15]), la creación de consultas sql dinámicas (*MyBatis* [23]), la creación y ejecución de pruebas automáticas (*Selenium* [24]), la creación y el manejo de plantillas XML (*JAXB* [26]), la parametrización de las pruebas (*TestNG* [25]), entre otros problemas enfrentados.

Personalmente gracias a este trabajo pude conocer por primera vez lo que significa la interacción con un cliente para el desarrollo o mantenimiento de un sistema. Específicamente el marco de trabajo usado para el desarrollo del nuevo módulo *Scrum* [11], demostró, **en mi opinión**, brindar las herramientas necesarias para guiar un proceso de manera adaptativa y con constante retroalimentación. Fue un poco más claro para mí el significado de la metodología ágil de desarrollo de software; personalmente pensaba que solo era una metodología que buscaba evadir todos los artefactos usados en las metodologías tradicionales que solo parecían ralentizar el proceso y que buscaba que el desarrollo se centrara más en el avance de la aplicación o sistema como tal por encima de la aplicación. Pero no es tan sencillo; la experiencia me enseñó que es un proceso más delicado, que no busca eliminar la documentación o los artefactos usados en las metodologías tradicionales, sino que solo se usen los necesarios; no busca acelerar el proceso de desarrollo de la aplicación, sino que busca agilizarlo, esto es que el cliente pueda ir evaluando la construcción de su producto e informando su visión de este a corto y largo plazo.

Otra facultad importante que he ido cultivando en mi vida profesional es el trabajo en equipo. El desarrollo de este trabajo me permitió aprender y ser testigo de las ventajas que conllevan el trabajo en equipo y que, en lugar de ser un obstáculo que merma la capacidad individual como al principio creía, es una forma de trabajo que potencia la calidad del resultado obtenido y la eficacia de su construcción, en este caso un producto de software.

4 TRABAJO FUTURO

El trabajo a futuro de este trabajo está relacionado directamente al sistema legado, y es su migración completa. La migración completa de un sistema es un proceso que lleva tiempo y esfuerzo, pero según el desarrollo de este trabajo se concluyó que, para este sistema específico, la migración de manera gradual del sistema es viable bajo ciertas condiciones específicas. En la medida que las condiciones sean cumplidas la migración del sistema puede ser llevada a cabo por el equipo de desarrollo.

Este trabajo implicaría la intervención de más personas ya que el sistema es grande y extenso en cuanto a la cantidad módulos que tiene y los procesos que lleva a cabo. Sin embargo, los resultados obtenidos fueron entregados a la compañía propietaria del sistema, y es ella quien finalmente decide si vale la pena o no llevar a cabo la migración completa de su sistema.

5 REFERENCIAS

[1] A. M. Langer, Guide to Software Development: Designing and Managing the Life Cycle, New York: springer, 2012, ISBN 1447122992, 9781447122999, p. 350.

[2] C. J. Shore, The Art of Agile Development, Usa: Usa: O'Reilly Media, Inc., 2008, ISBN 059655379X, 9780596553791.

[3] Oracle, «Oracle - Java,» 01 Junio 2014. [En línea]. Available: http://www.oracle.com/us/technologies/java/overview/index.html.

[4] C. C. Weber, «Build Security In,» [En línea]. Available: https://buildsecurityin.us-cert.gov/articles/best-practices/legacy-systems/assessing-security-risk-in-legacy-systems. [Último acceso: 01 Junio 2014].

[5] R. H. carlos Matos, «Migrating Legacy Systems to Service-Oriented Architectures,» *Electronic Communications of the EASST (European Association of Software Science and Technology),* vol. XVI, nº 1, p. 15, 2009.

[6] M. J. Escalona, M. Mejías, J. J. Gutiérrez y J. Torres, «MÉTODOS DE TESTING SOBRE LA INGENIERÍA DE REQUISITOS WEB,» *Conferencia Iberoamericana IADIS,* p. 8, 2004.

[7] A. Holmes y M. Kellogg, «Automating Functional Tests Using Selenium,» de *Proceedings of AGILE 2006 Conference (AGILE'06)*, Minneapolis, MN, 2006.

[8] A. G. Shaief, «Critical Success Factors for the Migration of Legacy Information Systems to SOA,» TUDelft, 2006, pp. 38-39.

[9] J. Debasish, de *JAVA AND OBJECT-ORIENTED PROGRAMMING PARADIGM*, New Delhi, PHI Learning Pvt. Ltd., 2005, pp. 06-11.

[10] W3C, «The Extensible Stylesheet Language Family (XSL),» [En línea]. Available: http://www.w3.org/Style/XSL/. [Último acceso: 01 Junio 2014].

[11] Scrum.org, «Scrum,» [En línea]. Available: https://www.scrum.org/. [Último acceso: 01 Junio 2014].

[12] T. A. S. Foundation, «Apache Maven Project,» [En línea]. Available: http://maven.apache.org/. [Último acceso: 01 Junio 2014].

[13] Checkstyle, «Checkstyle 5.7,» [En línea]. Available: http://checkstyle.sourceforge.net/. [Último acceso: 01 Junio 2014].

[14] S. SA, «Sun checks,» [En línea]. Available: https://dev.eclipse.org/sonar/rules_configuration/index/3. [Último acceso: 01 Junio 2014].

[15] P. S. Inc., «Spring Framework,» [En línea]. Available: http://projects.spring.io/spring-framework/. [Último acceso: 01 Junio 2014].

[16] MyBatis.org, «MyBatis,» [En línea]. Available: http://mybatis.github.io/mybatis-3/es/. [Último acceso: 01 Junio 2014].

[17] F. project, «FreeMarker - Overview,» [En línea]. Available: http://freemarker.org/. [Último acceso: 01 Junio 2014].

[18] F. Project, «FreeMarker vs. XSLT,» [En línea]. Available: http://freemarker.org/fmVsXSLT.html. [Último acceso: 01 Junio 2014].

[19] JUnit.org, «JUnit,» [En línea]. Available: http://junit.org/. [Último acceso: 01 Junio 2014].

[20] Oracle, «Working with URLs,» [En línea]. Available: http://docs.oracle.com/javase/tutorial/networking/urls/definition.html. [Último acceso: 01 Junio 2014].

[21] J. Touch, E. Lear, A. Mankin, M. Kojo, K. Ono, M. Stiemerling, L. Eggert, A. Melnikov, W. Eddy, E. Kohler y A. Mankin, «Service Name and Transport Protocol Port Number Registry,» [En línea]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt. [Último acceso: 01 Junio 2014].

[22] Oracle, «Java Api 7 - Interface Filter,» [En línea]. Available: http://docs.oracle.com/javaee/7/api/javax/servlet/Filter.html. [Último acceso: 01 Junio 2014].

[23] MyBatis, «SQL Dinámico,» [En línea]. Available: http://mybatis.github.io/mybatis-3/es/dynamic-sql.html. [Último acceso: 01 Junio 2014].

[24] S. C. [http://docs.seleniumhq.org/about/contributors.jsp], «Selenium Browser Automation,» [En línea]. Available: http://docs.seleniumhq.org/. [Último acceso: 01 Junio 2014].

[25] C. Beust, «TestNG,» [En línea]. Available: http://testng.org/doc/index.html. [Último acceso: 01 Junio 2014].

[26] Java.Net, «JAXB,» [En línea]. Available: https://jaxb.java.net/. [Último acceso: 01 Junio 2014].

6 ANEXOS

Como se aclara al inicio de este trabajo, el contenido de este documento corresponde a la parte teórica del desarrollo de los objetivos, la prueba de concepto es una implementación que ilustra el trabajo realizado en el transcurso de la pasantía; sus archivos fuentes y su manual de instalación se encuentran anexos a este documento.

6.1 Prueba de concepto - Manual de instalación

Contiene las instrucciones para instalar la prueba de concepto y poder ejecutar los proyectos que ahí se presentan y analizarlos de manera más detallada.

6.2 Prueba de concepto

Contiene los archivos fuentes de los proyectos que componen la prueba de concepto.